

Programando Excel VBA para Leigos \(PDFDrive.com

WALKENBACH , JOHN

VISIT...

LANZAROTE
Caliente.COM

Programando Excel VBA para Leigos \(PDFDrive.com

WALKENBACH , JOHN



Acelere na faixa de

Computadores/Spreadsheets

Tornando tudo mais fácil!

Tradução da 2ª Edição

alta velocidade do

Abra este livro e

P

descubra como:

r

Excel e dispare pelas

og

• **TrabalharnoVisualBasic**

Editor

personalizações de

ra

planilhas

ma

Programando

• **Dicasparalidarcomerrore**

exterminaçãodebugs

• **Maneirasdemontar**

Você está pronto para fazer o Excel pular

procedimentoseeventos

entre argolas? As suas planilhas funcionam

n

exatamente da maneira desejada? Com este

do

Excel® VBA

automáticos

• **Com o uso do gravador de**

livro, você pode dizer “sim”! Em seu interior,

macro Excel

você encontrará instruções passo a passo sobre

E

• **Técnicas para manipular**

como aperfeiçoar significativamente a potência

x

UserForm

do Excel, usando a linguagem VBA. Logo será

c

possível montar os seus próprios aplicativos

• **Instruções para projetar**

e

macros VBA

Excel para conseguir o máximo de seus dados.

l®

• **Etapas para acessar suas**

V

macrosatravésdainterface

- **Mergulhe em VBA** – veja como VBA é

B

deusuário

implementado em Excel, aplique técnicas avançadas

A

- **Segredosparacriar**

e use VBA com outros aplicativos Office

add-insExcel

- **Entre nele** – explore os conceitos de programação a partir de objetos Range e funções de planilhas

para eventos e manuseio de erros

Acesse o site

Aprenda a:

- **Comunicação é a chave** – aprenda como montar www.paraleigos.com.br

e conheça outros títulos!

- **Usar as ferramentas essenciais e as**

caixas de diálogo que parecem ter vindo

diretamente do laboratório de software da Microsoft

operações em VBA

- **Trabalhar com objetos Range e**

- **Segurança em primeiro lugar** – use macros Excel **fluxo de controle de programa**

para tornar as suas planilhas mais seguras e

bloquear vírus

- **Lidar com erros e eliminar**

bugs em seu código

- **Torne-o seu** – desenvolva funções personalizadas,
- **Desenvolver interfaces de usuário**

crie add-ins, projete aplicativos orientados por

personalizadas para os seus aplicativos,

usuário e incremente suas opções de carreira

Walkenbach

incluindo caixas de diálogo

www.altabooks.com.br

ISBN 978-85-7608-680-2

John Walkenbach também conhecido como “Mr.

JOHN WALKENBACH

Planilha” é a maior autoridade em Excel. Criador do premiado Power Utility Pack, Walkenbach escreveu

Autor do best-seller

mais de 50 livros e 300 artigos para publicações,

Programando Excel® 2007 VBA Para Leigos

incluindo PC World, InfoWorld e Windows.

VBA Excel.indd 1

15/02/2013 14:17:41

Programando Excel® VBA,

Tradução da 2ª Edição

Folha

de Cola

Funções

Abs

Retorna a um valor de um número absoluto

Exp

Retorna para o dia e a hora que o arquivo foi modificado pela ultima vez

Array

Retorna uma variante que esta em ordem

FileLen

Retorna para o número de bytes de um arquivo

Asc

Converte um caractere para um valor ASCII

Filter

Retorna um subconjunto de grande variedade que filtra base de Atn

Retorna o número inicial

critérios

CallByName

Invoca ou prepara um método de propriedade

Fix

Retorna a porção inteira de um número

CBool

Converte uma expressão para boolean

Format

Mostra uma expressão particularmente formatada

CByte

Converte uma expressão para um tipo de dados

Format

Retorna um número como uma linha formatada como moeda

Currency

CCur

Converte uma expressão de um tipo de dados para a moeda

CDate Converte uma expressão de um tipo de dados para o dia

FormatDate

Retorna um número como uma linha formatada no dia e hora

Time

CDBl

Converte uma expressão de um tipo de dados em dupla

Format

Retorna um número formatado como linha

CDec

Converte uma expressão de um tipo de dados em

Number

Choose

Seleciona e retorna um valor da lista

Format

Retorna um número como uma linha, formatado com

Percent

porcentagem

Chr

Converte um valor ANSI da fila

FreeFile

Retorna para o próximo arquivo disponível para usar a

CInt

Converte um tipo de dados em uma expressão inteira

declaração Open

CLng

Converte uma expressão longa de um tipo de dados

FV

Retorna o futuro valor baseado no período de anuidade fixa,
pagamentos e taxa de juros fixa

Cos

Retorna o co-seno de um número

GetAll

Retorna uma lista de configurações e valores (configuração original
CreateObject

Cria um automaticamente um objeto OLE

criada com SaveSetting) de uma aplicação registrada pelo Windows
CSng

Converte um tipo de dados em uma única expressão

GetAttr

Retorna um código atribuído que representa um arquivo

CStr

Converte uma fila de expressão de um tipo de dados

GetObject

Recupera um objeto OLE de um arquivo

CurDir

Retorna par o caminho atual

GetSetting a

Retorna um valor de uma chave de configuração registrada pelo Windows

CVar

Converte um tipo de dados em uma variável

Hex

Converte decimal em hexadecimal

CVDate

Converte um tipo de dados um uma expressão

Hour

Retorna o tempo e hora

CVErr

Retorna até um número definido como errado

IIf

Retorna um em duas partes, dependendo da avaliação de uma

Date

Retorna até o sistema de dados atual

expressão

DateAdd

Retorna a uma data especifica acrescentando intervalos

Input

Retorna um número específico de números de caracteres para sequenciais

um arquivo de texto

DateDiff

Retorna a uma data especifica subtraindo intervalos

InputB

Retorna um número específico de bytes para um arquivo de texto

DatePart

Retorna a uma parte da data que contém algo específico

InputBox

Mostra uma caixa de entrada do prompt para o usuário

DateSerial

Converte uma data em uma serie de número

InStr

Retorna a posição de uma linha para dentro de uma outra linha
DateValue

Converte um fila de dados

InStrB

Retorna a posição do byte de uma linha para dentro de uma outra Day

Retorna o dia do mês de um dado

linha

DDB

Retorna um ativo para um período específico usando um método

InStrRev

Retorna a posição de uma linha dentro de uma outra , começam—
decrecente

do do final da linha

Dir

Retorna o nome de um arquivo ou diretório correspondente a

Int

Retorna a parte inteira de um número

um padrão

IPmt

Retorna o pagamento de juros dando o período de anuidade fixado
DoEvents

Submete a execução, para que o sistema possa processar outros
eventos

IRR

Retorna taxa interna para um período de fluxo de dinheiro

Environ

Retorna uma fila associada com o sistema de operação variável
IsArray

Retorna True se a variável estiver em ordem

EOF

Retorna True se o texto do arquivo tiver chego no final

IsDate

Retorna True se a variável for data

Error

Retorna a mensagem de erro correspondente a um número

IsEmpty

Retorna True se a variável for iniciada

incorreto

IsError

Retorna True se uma expressão para um valor incorreto

Exp

Retorna a base natural do logarítimo aumentando a força

IsMissing

Retorna True se um argumento opcional não foi

FileAttr

Retorna o modo de arquivo para texto

Para Leigos: a série de livros para iniciantes que mais vende no mundo.

ExcelVBABokks.indb 1

15/02/2013 13:59:14

Programando Excel® VBA,

Folha

Tradução da 2ª Edição

de Cola

IsNumeric

Retorna True se uma expressão pode ser avaliada como um

Spc

Posiciona o output em um fluxo output

número

Split

Retorna uma ordem constante de números de uma sublinha

IsObject

Retorna True se uma expressão referir um objeto automático

OLE

Sqr

Retorna à raiz quadrada de um número

Join

Retorna uma linha criada por um número ligado a uma sublinha

Str

Retorna uma linha representada por um número

contida em uma ordem

Right

Retorna um número específico de caracteres do lado direito

LBound

Retorna uma ordem de um nível inferior

de uma linha

LCase

Retorna uma linha convertida em minúscula

RightB

Retorna um número específico de bytes do lado direito de uma

linha

Left

Retorna um específico caractere de número do lado esquerdo

da linha

Rnd

Retorna um número aleatório entre 0 e 1

LeftB

Retorna um específico número de bytes do lado esquerdo da

Round

Rodeia um número para um específico espaço de número

linha

decimal

Len

Retorna quantidade de uma linha em caracteres

RTrim

Retorna uma copia de uma linha sem espaços rasteiros

LenB

Retorna a quantidade de uma linha em bytes

Second

Retorna o segundo

Loc

Retorna a atual posição de leitura ou escrita do texto

Seek

Retorna a atual posição do arquivo de texto

LOF

Retorna um número de bytes em arquivo de texto

Sgn

Retorna um inteiro que indica o sinal de um número

Log

Retorna o logaritmo natural de um número

Shell

Roda um programa executável

LTrim

Retorna a cópia de uma linha sem espaço principal

Sin

Retorna a um número determinado

Mid

Retorna um número específico de um caractere de uma linha

StrComp

Retorna um valor indicado, resultado de uma comparação

MidB

Retorna um número específico de bytes de uma linha

StrConv

Retorna a variante de uma linha

Minute

Retorna o minuto de uma hora desejada

String

Retorna a uma linha ou caractere repetidos

MIRR

Retorna a taxa interna para um período de fluxo de dinheiro StrRe-
Reverte a ordem de uma sequência de caracteres

(usando diferentes taxas)

verse

Month

Retorna para o mês de um dia desejado

Switch

Avalia uma lista de expressões e retorna o valor associado com a primeira expressão da lista que é True

Month—

Retorna uma linha indicando o específico mês

Name

SYD

Retorna a soma de anos depreciada em um determinado período

MsgBox

Mostra a forma de caixa de mensagens

Tab

Posiciona o output em um fluxo output

Now

Retorna ao sistema atual de data e hora

Tan

Retorna a tangente de um número

NPer

Retorna a um número de anuidade baseada no período fixo de pagamentos e taxa de juros

Time

Retorna ao tempo atual do sistema

NPV

Retorna o valor líquido atual de um investimento baseado no

Timer

Retorna o número de um Segundo desde a meia-noite

período de fluxo de dinheiro e desconto de taxa

TimeSe—

Retorna o tempo especificando a hora minuto e segundo

Oct

Converte decimal em octal

rial

Partition

Retorna uma variante de linha indicando onde ocorre o número

TimeVa—

Converte uma linha para uma serie de números

calculado em uma serie de colunas

lue

Pmt

Retorna o pagamento para a anuidade baseada em um período,

Trim

Retorna a linha contendo uma copia da linha especificada sem

fixo e taxa de juros fixa

limite de espaços

PPmt

Retorna o principal pagamento para dar um período de anuidade

TypeNa-Retorna uma linha que descreve o tipo de dado da variável

baseado na taxa de juros fixa

me

PV

Retorna o presente valor de uma anuidade baseado no período

UBound

Retorna uma ordem superior

fixo de pagamento, para ser pago no futuro e fixado a taxa de juros

UCase

Converte uma linha em uppercase

QBColor

Retorna o RGB correspondente à cor de um número específico

Val

Retorna os números contidos em uma linha

(compativelmente usado no Quick Basic)

VarType

Retorna o valor indicando o subtipo de variável

Rate

Retorna a taxa de juros por período de anuidade

Weekday

Retorna um número representando o dia da semana

Replace

Retorna uma linha onde uma sublinha foi repostada

Weekday

Retorna uma linha indicando o dia da semana específico

RGB

Retorna um número representando um valor colorido RGB

Year

Retorna o ano da data

Space

Retorna uma linha com um específico número de espaços

Para Leigos: a série de Livros para iniciantes que mais vende no mundo.

ExcelVBABokks.indb 2

04/02/2013 14:01:35

Programando

Excel® VBA

Tradução da 2ª Edição

John Walkenbach

Rio de Janeiro, 2013

ExcelVBABokks.indb 1

15/02/2013 13:59:15

Programando Excel® VBA Para Leigos, Tradução da 2ª Edição
Copyright © 2013 da Starlin Alta Editora e Consultoria Eireli.

ISBN: 978-85-7608-680-2

Translated From Original: Excel VBA Programming For Dummies
ISBN: 978-0-470-50369-0. Original English language edition
Copyright © 2007 by Wiley Publishing, Inc. All rights reserved
including the right of reproduction in whole or in part in any form.
This translation published by arrangement with Wiley Publishing, Inc.
Portuguese language edition Copyright © 2013 by Starlin Alta Editora
e Consultoria Eireli. All rights reserved including the right of
reproduction in whole or in part in any form.

“Wiley, the Wiley Publishing Logo, for Dummies, the Dummies Man and related trad dress are trademarks or registered trademarks of John Wiley and Sons, Inc. and/or its affiliates in the United States and/or other

countries. Used under license.

Todos os direitos reservados e protegidos por Lei. Nenhuma parte deste livro, sem autorização prévia por escrito da editora, poderá ser repro-duzida ou transmitida.

Erratas: No site da editora relatamos, com a devida correção, qualquer erro encontrado em nossos livros.

Marcas Registradas: Todos os termos mencionados e reconhecidos como Marca Registrada e/ou Comercial são de responsabilidade de seus proprietários. A Editora informa não estar associada a nenhum produto e/ou fornecedor apresentado no livro.

Impresso no Brasil

Vedada, nos termos da lei, a reprodução total ou parcial deste livro.

Produção Editorial

Equipe de Design

Tradução

Editora Alta Books

Adalberto Taconi

Juliana Morais Missina

Gerência Editorial

Bruna Serrano

Copidesque

Anderson Vieira

Iuri Santos

Marco Aurélio Silva

Savannah Hartman

Supervisão Gráfica

e Editorial

Equipe Editorial

Revisão Gramatical

Angel Cabeza

Ana Lucia Silva

Equipe Alta Books

Supervisão de

Brenda Ramalho

Revisão Gramatical

Qualidade Editorial

Camila Werhahn

Jefferson Santos

Sergio Luiz de Souza

Claudia Braga

Daniel Siqueira

Diagramação

Conselho de

Evellyn Pacheco

Francisca Santos

Qualidade Editorial

Jaciara Lima

Adalberto Taconi

Juliana de Paulo

Marketing e Promoção

Anderson Vieira

Licia Oliveira

Daniel Schilklaper

Angel Cabeza

Milena Souza

marketing@altabooks.com.br

Pedro Sá

Natália Gonçalves

Sergio Luiz de Souza

Paulo Camerino

Editoria Atualização

Rafael Surgek

Augusto Coutinho

Thiê Alves

Cristiane Santos

Vinicius Damasceno

Marcelo Vieira

1ª reimpressão, outubro de 2013

Vanessa Gomes

Dados Internacionais de Catalogação na Publicação (CIP)

W177p Walkenbach, John.

Programando Excel VBA para leigos / John Walkenbach. – Rio de Janeiro, RJ : Alta Books, 2012.

408 p. : il. ; 24 cm. – (Para leigos)

Inclui índice.

Tradução de: Excel VBA programming for dummies (2. ed.).

ISBN 978-85-7608-680-2

1. Excel (Programa de computador). 2. Visual Basic for applications (Linguagem de programação de computador). 3. Planilhas eletrônicas. 4. Negócios - Programas de computador. I. Título. II. Série.

CDU 004.4:004.438

CDD 005.3

Índice para catálogo sistemático:

1. Excel (Programa de computador) : Visual Basic for applications
004.4:004.438

2. Visual Basic for applications : Excel (Programa de computador)
004.4:004.438

(Bibliotecária responsável: Sabrina Leal Araujo – CRB 10/1507) Rua
Viúva Cláudio, 291 – Bairro Industrial do Jacaré

CEP: 20970-031 – Rio de Janeiro – Tels.: 21 3278-8069/8419 Fax: 21
3277-1253

www.altabooks.com.br – e-mail: altabooks@altabooks.com.br

www.facebook.com/altabooks – www.twitter.com/alta_books

ExcelVBABokks.indb 2

15/02/2013 13:59:15

Sobre o Autor

John Malkenbach é o autor *best-seller* de mais de 50 livros de planilhas e mora no sul do Arizona. Quando ele não está usando Excel, provavelmente está tocando banjo, trajando casaca.

ExcelVBABokks.indb 3

15/02/2013 13:59:15

Dedicatória

Este livro é dedicado ao Toad e à Sra. Toad porque estou ficando sem

pessoas para dedicatórias em livro.

Agradecimentos do Autor

Eu sou grato a todas as pessoas da Wiley Publishing por me dar a oportunidade de escrever livros sobre Excel. Agradecimentos especiais a Colleen Tottz Diamond (editor de projeto) e a Jan Karel Pieterse (editor técnico). Com certeza eles tornaram o meu trabalho muito mais fácil.

ExcelVBABokks.indb 4

15/02/2013 13:59:15

Sumário Resumido

<i>Introdução</i>	<i>1</i>
<i>Parte I: Introdução ao VBA</i>	<i>11</i>
Capítulo 1: O Que É VBA ?	13
Capítulo 2: Saltando Para o Lugar Certo	23
<i>Parte II: Como o VBA Trabalha com o Excel</i>	<i>35</i>
Capítulo 3: Trabalhando no Visual Basic Editor	37
Capítulo 4: Introdução ao Modelo de Objeto do Excel.....	55
Capítulo 5: Procedimentos Function e Sub no VBA	69
Capítulo 6: Usando o Gravador de Macro do Excel	81
<i>Parte III: Conceitos de Programação</i>	<i>93</i>
Capítulo 7: Elementos Essenciais da Linguagem VBA	95
Capítulo 8: Trabalhando com Objetos Range	115

Capítulo 9: Usando VBA e Funções de Planilha	129
Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões	141
Capítulo 11: Procedimentos e Eventos Automáticos	161
Capítulo 12: Técnicas de Tratamento de Erros	183
Capítulo 13: Técnicas de Extermínio de Bugs	197
Capítulo 14: Exemplos de Programação em VBA	211
<i>Parte IV: Como se Comunicar com Seus Usuários</i>	<i>233</i>
Capítulo 15: Caixas de Diálogo Simples	235
Capítulo 16: Princípios Básicos de UserForm	253
Capítulo 17: Usando os Controles de UserForm	269
Capítulo 18: Técnicas e Truques do UserForm	289
Capítulo 19: Como Acessar suas Macros através da Interface de Usuário	313
ExcelVBABokks.indb	5
15/02/2013 13:59:15	
vi Programando Excel VBA Para Leigos, Tradução da 2ª Edição	
<i>Parte V: Juntando Tudo</i>	<i>331</i>
Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar	333
Capítulo 21: Criando Add-Ins do Excel.....	349

Parte VI: A Parte dos Dez 361

Capítulo 22: Dez Perguntas de VBA (E Respostas)
..... 363

Capítulo 23: (Quase) Dez Recursos do Excel
..... 367

Índice371

ExcelVBABokks.indb 6

15/02/2013 13:59:15

Sumário

Introdução 1

É Este o Livro Certo?
.....1

Então, Você Quer Ser um Programador...
.....2

Por Que se Preocupar?
.....3

O Que Presumo Sobre Você
.....3

Seção Obrigatória das Convenções
Tipográficas.....4

Verifique Suas Configurações de Segurança
.....5

Como Este Livro Está Organizado
.....6

Parte I: Introdução ao VBA
.....6

Parte II: Como VBA Funciona com Excel
.....6

Parte III: Conceitos de Programação
.....7

Parte IV: Comunicação com Seus	
Usuários.....	7

Parte V: Juntando Tudo	
.....	7

Parte VI: A Parte dos Dez	
.....	7

Espere, Há Mais!	
.....	7

Ícones Usados Neste Livro	
.....	7

Como Obter os Arquivos de Exemplos	
.....	8

E Agora?	
.....	

<i>Parte I: Introdução ao VBA</i>	11
---	----

Capítulo 1: O Que É VBA ?

.....

Tudo Bem, Então o Que é VBA?	
.....	13

O Que Você Pode Fazer com VBA?	
.....	14

Inserir um grupo de texto	
.....	15

Automatizar tarefas executadas com frequência	
.....	15

Automatizar operações repetitivas	
.....	15

Criar um comando personalizado	
.....	15

Criar um botão personalizado	
.....	16

Desenvolver novas funções de planilhas	16
Criar aplicativos completos, guiados por macro	16
Criar suplementos (add-ins) personalizados para o Excel	16
Vantagens e Desvantagens do VBA	16
ExcelVBABokks.indb 7	
15/02/2013 13:59:15	
viii Programando Excel VBA Para Leigos, Tradução da 2ª Edição	
Vantagens do VBA	17
Desvantagens do VBA	17
VBA Resumidamente	18
Uma Excursão pelas Versões Anteriores	20
Capítulo 2: Saltando Para o Lugar Certo	23
Primeiros Passos	23
Usuários de Excel 2010	24
Usuários de Excel 2007	24
O Que Você Fará	25
Dando os Primeiros Passos	25

Gravando a Macro	26
Testando a Macro	27
Examinando a Macro	27
Modificando a Macro	30
Salvando Planilhas que Contêm Macros	30
Entendendo a Segurança de Macro	31
Mais sobre a Macro NameAndTime	33
<i>Parte II: Como o VBA Trabalha com o Excel</i>	<i>35</i>
Capítulo 3: Trabalhando no Visual Basic Editor	37
O Que É o Visual Basic Editor?	37
Ativando o VBE	37
Entendendo os componentes do VBE	38
Como Trabalhar com a Janela de Projeto	40
Adicionando um novo módulo VBA	41
Removendo um módulo VBA	41
Exportando e importando objetos	42

Trabalhando com a Janela de Código	42
Minimizando e maximizando janelas	42
Criando um módulo	44
Como inserir código VBA em um módulo	44
Inserindo o código diretamente	45
Usando o gravador de macro	47
Copiando o código VBA	49
Personalizando o Ambiente VBA	49
Usando a guia Editor	50
Usando a guia Formato do editor	52
ExcelVBABokks.indb	8
15/02/2013 13:59:15	

Sumário

ix

Usando a guia Geral	53
Usando a guia Encaixe	54

Capítulo 4: Introdução ao Modelo de Objeto do Excel

.....	55
Excel É um Objeto?	56
.....	56
Escalando a Hierarquia de Objetos	56
.....	56
Enchendo Sua Cabeça com Coleções	58
.....	58
Como Fazer Referência aos Objetos	58
.....	58
Como navegar pela hierarquia	59
.....	59
Simplificando referências a objetos	60
.....	60
Mergulhando nas Propriedades e nos Métodos do Objeto	60
.....	60
Propriedades do objeto	62
.....	62
Métodos de Objeto.....	63
Eventos de objeto.....	64
Descobrimos Mais	64
.....	64
Usando o sistema de Ajuda de VBA	65
.....	65
Como usar o Pesquisador de Objeto	66
.....	66
Como relacionar automaticamente propriedades e métodos	66
.....	66
Capítulo 5: Procedimentos Function e Sub no VBA	69
.....	69

Subs versus Funções	69
Observando os procedimentos Sub	70
Observando os procedimentos Function	70
Nomeando Subs e Functions	71
Executando Procedimentos Sub	71
Executando diretamente o procedimento Sub	73
Executando um procedimento a partir da caixa de diálogo Macro	74
Executando uma macro usando uma tecla atalho	75
Executando um procedimento a partir de um botão ou forma	76
Executando um procedimento a partir de outro procedimento	78
Executando Procedimentos Function	78
Chamando uma função a partir de um procedimento Sub	79
Chamando uma função a partir de uma fórmula de planilha	79
Capítulo 6: Usando o Gravador de Macro do Excel	81
Isto Está Vivo ou É VBA?	81
O Básico sobre Gravação	82

Preparação para Gravar	84
Relativo ou Absoluto?	84
ExcelVBABokks.indb 9	
15/02/2013 13:59:15	
x Programando Excel VBA Para Leigos, Tradução da 2ª Edição	
Gravando no modo absoluto	84
Gravando no modo relativo	85
O Que É Gravado?	87
Opções da Gravação	88
Nome da Macro	88
Tecla de Atalho	89
Armazenar Macro Em	89
Descrição	89
Essa Coisa É Eficiente?	90
<i>Parte III: Conceitos de Programação</i>	93
Capítulo 7: Elementos Essenciais da Linguagem VBA	95
Usando Comentários em Seu Código VBA	95
Usando Variáveis, Constantes e Tipos de Dados	

.....	97
Entendendo variáveis	97
.....	97
O que são tipos de dados do VBA?	98
.....	98
Declarando e estendendo variáveis	99
.....	99
Trabalhando com constantes	105
Constantes pré-fabricadas	106
.....	106
Trabalhando com strings	107
.....	107
Trabalhando com datas	108
.....	108
Usando Declarações de Atribuição	109
.....	109
Exemplos de declaração de atribuição	109
.....	109
Sobre aquele sinal de igual	109
.....	109
Operadores regulares	110
.....	110
Trabalhando com Arrays	112
.....	112
Declarando arrays	112
.....	112
Arrays multidimensionais	113
.....	113
Arrays dinâmicos	113
.....	113

Usando Labels (Etiquetas)	114
Capítulo 8: Trabalhando com Objetos Range	115
Uma Revisão Rápida	115
Outras Maneiras de Fazer Referência a uma Faixa	117
A propriedade Cells	117
A propriedade Offset	118
Fazendo referência a colunas e linhas inteiras	119
Algumas Propriedades úteis do Objeto Range	119
ExcelVBABokks.indb 10	
15/02/2013 13:59:15	

Sumário

<i>xi</i>	
A propriedade Value	120
A propriedade Text	121
A propriedade Count	121
As propriedades Column e Row	121
A propriedade Address	

.....	122
A propriedade	
HasFormula.....	122
A propriedade Font	
.....	123
A propriedade Interior	
.....	123
A propriedade Formula	
.....	124
A propriedade NumberFormat	
.....	125
Alguns Métodos úteis do Objeto Range	
.....	126
O método Select	
.....	126
Os métodos Copy e Paste	
.....	127
O método Clear	
.....	127
O método Delete	
.....	128
Capítulo 9: Usando VBA e Funções de Planilha	
.....	129
O Que É uma Função?	
.....	129
Usando Funções VBA Integradas	
.....	130
Exemplo de função VBA	
.....	130
Funções VBA que fazem mais do que retornar um valor	
.....	132

Descobrimos funções VBA	133
Usando Funções de Planilha no VBA	135
Exemplos de funções e planilhas	136
Introduzindo funções de planilha	138
Mais Sobre o Uso de Funções de Planilha	139
Usando Funções Personalizadas	139
Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões	141
Seguindo o Fluxo, Cara	141
A Declaração GoTo	142
Decisões, decisões	143
A estrutura If-Then	143
A estrutura Select Case	148
Fazendo Seu Código Dar um Loop	151
Loop For-Next	152
Loop DoWhile	156
Loop DoUntil	157

Fazendo Loop através de uma Collection	158
ExcelVBABokks.indb 11	
15/02/2013 13:59:16	
xii Programando Excel VBA Para Leigos, Tradução da 2ª Edição	
Capítulo 11: Procedimentos e Eventos Automáticos	161
Preparação para o Grande Evento	161
Os eventos são úteis?	163
Programando procedimentos que lidam com eventos	164
Aonde Vai o Código VBA?	164
Escrevendo um Procedimento Que Lida com Evento	165
Exemplos Introdutórios	167
O evento Open para uma pasta de trabalho	167
O evento BeforeClose para uma pasta de trabalho	169
O evento BeforeSave para uma pasta de trabalho	169
Exemplos de Ativação de Eventos	170
Ativar e desativar eventos em uma planilha	170
Ativar e desativar eventos em uma pasta de trabalho	171

Eventos de ativação de pasta de trabalho	173
Outros Eventos Relacionados à Worksheet (Planilha)	174
O evento BeforeDoubleClick	174
O evento BeforeRightClick	174
O evento Change	175
Eventos Não Associados a Objetos	177
O evento OnTime	178
Eventos de pressionamento de teclas	180
Capítulo 12: Técnicas de Tratamento de Erros	183
Tipos de Erros.....	183
Um Exemplo Errôneo	184
A imperfeição da macro	185
A macro ainda não é perfeita.....	186
A macro já está perfeita?	186
Desistindo da perfeição	187
Como Lidar com Erros de Outra Maneira	188

Reverendo o procedimento EnterSquareRoot	188
Sobre a declaração On Error	189
Como Lidar com Erros: Os Detalhes	190
Recuperação depois de um erro	190
Lidando com erros resumidamente	192
Como saber quando ignorar erros	192
Como identificar erros específicos	193
Um Erro Intencional	194
ExcelVBABokks.indb 12	
15/02/2013 13:59:16	

Sumário

xiii

Capítulo 13: Técnicas de Extermínio de Bugs 197

Espécies de Bugs	197
Como Identificar Bugs	198
Técnicas de Depuração	199
Como examinar o seu código	

.....	199
Usando a função MsgBox	200
.....	200
Inserindo declarações Debug.Print	201
.....	201
Usando o depurador VBA	202
.....	202
Sobre o Depurador	202
.....	202
Configurando pontos de interrupção em seu código	202
.....	202
Usando a janela Inspeção de Variáveis	206
.....	206
Usando a janela de Variáveis locais	207
.....	207
Dicas para Redução de Bugs	208
.....	208
Capítulo 14: Exemplos de Programação em VBA	211
.....	211
Como Trabalhar com Ranges (faixas)	211
.....	211
Copiando uma faixa	212
.....	212
Copiando uma faixa de tamanho variável	213
.....	213
Selecionando ao final de uma linha ou coluna	214
.....	214
Selecionando uma linha ou coluna	215
.....	215
Movendo uma faixa	215
.....	215

Como fazer loop eficientemente através de uma faixa	216
Como fazer loop eficientemente através de uma faixa (Parte II)	217
Solicitando o valor de uma célula	218
Determinando o tipo de seleção	219
Identificando uma seleção múltipla	219
Mudando as Configurações do Excel	220
Mudando configurações Booleanas	221
Mudando configurações não Booleanas.....	221
Trabalhando com Gráficos	222
Modificando o tipo de gráfico	224
Fazendo Looping através da coleção ChartObjects	224
Modificando propriedades Chart	225
Aplicando formatação de gráfico.....	225
Dicas de Velocidade do VBA	227
Desativando a atualização de tela	227
Desativando o cálculo automático	228

Eliminando aquelas inoportunas mensagens de alerta	228
ExcelVBABokks.indb 13	
15/02/2013 13:59:16	
xiv Programando Excel VBA Para Leigos, Tradução da 2ª Edição Simplificando referências de objeto	229
Declarando tipos de variáveis	230
Como usar a estrutura With-End With	231
<i>Parte IV: Como se Comunicar com Seus Usuários</i>	233
Capítulo 15: Caixas de Diálogo Simples	235
Alternativas a UserForm	235
A Função MsgBox	236
Obtendo uma resposta de uma caixa de mensagem	237
Personalizando caixas de mensagem	238
A Função InputBox	241
Sintaxe InputBox	241
Um exemplo de InputBox	242
O Método GetOpenFilename	244
A sintaxe para o método GetOpenFilename	

.....	244
Um exemplo de GetOpenFilename	245
.....	245
Selecionando múltiplos arquivos	247
.....	247
O Método GetSaveAsFileName	248
.....	248
Como Obter um Nome de Pasta	249
.....	249
Exibindo as Caixas de Diálogo Integradas do Excel.....	250
Capítulo 16: Princípios Básicos de UserForm	253
.....	253
Como Saber Quando Usar um UserForm	253
.....	253
Criando UserForms: Uma Visão Geral	254
.....	254
Trabalhando com UserForms.....	255
Inserindo um novo UserForm	255
.....	255
Adicionando controles a um UserForm	256
.....	256
Mudando propriedades em um controle UserForm	257
.....	257
Observando a janela de Código de UserForm	258
.....	258
Exibindo um UserForm	259
.....	259
Usando informações de um UserForm	259
.....	259

Um Exemplo de UserForm	260
Criando o UserForm	260
Adicionando os botões de comando	261
Adicionando os botões de opção	262
Adicionando procedimentos que lidam com eventos	263
Criando uma macro para exibir a caixa de diálogo	265
Como disponibilizar a macro	266
Testando a macro	267
ExcelVBABokks.indb 14	
15/02/2013 13:59:16	

Sumário

xv

Capítulo 17: Usando os Controles de UserForm 269

Começando com os Controles da Caixa de Diálogo	269
Adicionando controles	269
Introduzindo propriedades de controle	270
Controles de Caixa de Diálogo: Os Detalhes	

.....	273
Controle Caixa de Seleção.....	274
Controle Caixa de Combinação.....	274
Controle Botão de comando.....	275
Controle Quadro.....	276
Controle Imagem.....	276
Controle Rótulo.....	277
Controle Caixa de Listagem.....	278
Controle Multipágina.....	279
Controle Botão de Opção.....	279
Controle RefEdit.....	280
Controle Barra de Rolagem.....	281
Controle Botão de Rotação.....	282
Controle TabStrip.....	282
Controle Caixa de Texto.....	282
Controle ToggleButton.....	283

Trabalhando com Controles de Caixa de Diálogo	284
Movendo e redimensionando controles	284
Alinhando e espaçando controles	284
Acomodando teclado de usuários	285
Testando um UserForm	287
Estética de Caixa de Diálogo	287
Capítulo 18: Técnicas e Truques do UserForm	289
Como Usar Caixas de Diálogo	289
Um Exemplo de UserForm	289
Criando a caixa de diálogo	290
Escrevendo código para exibir a caixa de diálogo	292
Disponibilizando a macro	292
Testando a sua caixa de diálogo	293
Adicionando procedimentos que lidam com eventos	294
Validando os dados	295
Agora a caixa de diálogo funciona	296

Mais Exemplos do UserForm	296
---------------------------	-----

Um exemplo de Caixa de Listagem	296
---------------------------------	-----

ExcelVBABokks.indb 15

15/02/2013 13:59:16

xvi Programando Excel VBA Para Leigos, Tradução da 2ª Edição	
Preenchendo uma Caixa de Listagem	297

Selecionando uma faixa	301
------------------------	-----

Usando múltiplos conjuntos de Botões de opção	302
---	-----

Utilizando um Botão de Rotação e uma Caixa de Texto	303
---	-----

Usando um UserForm como um indicador de progresso	305
---	-----

Criação de uma caixa de diálogo Multipágina	308
---	-----

Exibindo um gráfico em um UserForm	310
------------------------------------	-----

Uma Lista de Verificação de Caixa de Diálogo	311
--	-----

Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 313

O Que Aconteceu com CommandBars?	313
----------------------------------	-----

Personalização da Faixa de Opções	314
-----------------------------------	-----

Como personalizar manualmente a Faixa de Opções	314
---	-----

Personalizando a Faixa de Opções com XML	316
Personalizando Menus de Atalho	321
Comandando a coleção de CommandBars	321
Listando todos os menus de atalho	321
Referência a CommandBars	322
Referência a controles em um CommandBar	323
Propriedades de controles CommandBar	324
Exemplos de Menu de Atalho VBA	326
Adicionando um novo item ao menu de atalho Cell	326
Desativando um menu de atalho	328
Criando uma Barra de Ferramentas Personalizadas	329
<i>Parte V: Juntando Tudo</i>	<i>331</i>
Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar	333
Por Que Criar Funções Personalizadas?	333
Como Entender os Princípios Básicos de Função VBA	334
Escrevendo Funções	335

Trabalhando com Argumentos de Função	335
Exemplos de Função	336
Uma função sem argumento	336
Uma função com um argumento	336
Uma função com dois argumentos	338
Uma função com um argumento faixa	339
Uma função com um argumento opcional	340
Uma função com um número indefinido de argumentos	342
ExcelVBABokks.indb	16
15/02/2013 13:59:16	

Sumário xvii

Funções Que Retornam um Array	343
Retornando um array de nomes de meses	343
Retornando uma lista classificada	344
Como Usar a Caixa de Diálogo Inserir Função	345
Exibindo a descrição da função	346

Descrições de argumento	347
Capítulo 21: Criando Add-Ins do Excel	349
Certo ... Então, o Que é um Add-In?	349
Por Que Criar Add-Ins?	350
Trabalhando com Add-Ins	351
Princípios Básicos do Add-In	352
Um Exemplo de Add-In	353
Configurando a pasta de trabalho	353
Testando a pasta de trabalho	355
Como adicionar informações descritivas	356
Protegendo o código VBA	357
Criando o add-in	357
Abrindo o add-in	357
Distribuindo o add-in	358
Como modificar o add-in	359
<i>Parte VI: A Parte dos Dez</i>	<i>361</i>

Capítulo 22: Dez Perguntas de VBA (E Respostas)

..... 363

Capítulo 23: (Quase) Dez Recursos do Excel

..... 367

O Sistema de Ajuda do VBA

.....367

Suporte de Produtos Microsoft

.....367

Grupos de Notícias da Internet

.....368

Sites da Internet

.....369

Blogs do Excel

.....369

Google

.....

Bing

.....

Grupos e Usuários Locais

.....370

Meus Outros Livros

.....370

Índice 371

ExcelVBABokks.indb 17

15/02/2013 13:59:16

xviii Programando Excel VBA Para Leigos, Tradução da 2ª Edição

ExcelVBABokks.indb 18

15/02/2013 13:59:16

Introdução





S saudações, futuro programador de Excel...

Obrigado por comprar este livro. Creio que você descobrirá que ele oferece uma maneira rápida e agradável de encontrar os prós e os contras de programação em Microsoft Excel. Ainda que você não tenha a mais vaga ideia do que se trata programação, este livro pode ajudá-lo a fazer o Excel pular através de argolas rapidamente (bem, talvez demore algum tempo).

Diferentemente de outros livros de programação, este foi escrito em linguagem simples, para que pessoas normais pudessem entender.

Melhor ainda, ele está cheio de informações do tipo “só os fatos” – e não do tipo que você poderia precisar uma vez a cada três gerações.

É Este o Livro Certo?

Vá até qualquer grande livreria e você descobrirá muitos livros sobre Excel (livros demais, na minha opinião). Uma rápida olhada pode ajudá-lo a decidir se este livro é realmente o certo para você. Este livro: 5 Foi desenvolvido para aqueles que pretendem se adaptar rapidamente à programação de Aplicativos Visual Basic (VBA).

5 Não requer experiência anterior com programação.

5 Cobre os comandos mais comuns.

5 É adequado para o Excel 2007 ou para o Excel 2010.

5 Você poderá até dar um sorriso ocasionalmente – ele tem até desenhos animados.

Se você está usando Excel 2000, XP, ou 2003, este livro não é para você.

O Excel 2007 e o Excel 2010 são muito diferentes das versões anteriores.

Caso ainda esteja usando uma versão de Excel anterior à de 2007, procure um livro que seja específico para aquela versão.

ExcelVBABokks.indb 1

15/02/2013 13:59:16

2

Programando Excel VBA Para Leigos, Tradução da 2a Edição

A propósito, este *não* é um livro de introdução ao Excel. Se você estiver procurando por um livro de Excel de objetivos gerais, verifique quaisquer outros livros publicados pela Alta Books, no site www.altabooks.com.br.

Note que o título deste livro não é *O Guia Completo de Programação VBA em Excel Para Leigos*. Eu não abordo todos os aspectos de programação em Excel — e, novamente, é provável que você não queira saber *tudo* sobre esse assunto. No caso, improvável, de querer um livro de programação de Excel mais compreensível, poderia experimentar o *Microsoft Excel 2010 Power Programming com VBA*, de John Walkenbach (esse camarada é prolífico ou o quê?), publicado pela Wiley. E sim, também está disponível uma edição para Excel 2007.

Então, Você Quer Ser um

Programador...

Além de ganhar dinheiro para pagar minhas contas, meu objetivo principal é mostrar para usuários de Excel como usar a linguagem VBA — uma ferramenta que ajuda a ampliar significativamente o poder da planilha mais popular do mundo. No entanto, usar VBA envolve programação.

Se você é como a maioria dos usuários de computador, a palavra *programador* cria uma imagem de alguém que se parece e se comporta de um modo totalmente diferente de você. Talvez palavras como *nerd*, *geek* e *dweeb* apareçam na memória.

Os tempos mudaram. Programar computadores se tornou mais fácil, e até as pessoas chamadas normais fazem isso — e até admitem isso aos amigos e familiares. *Programar* significa apenas desenvolver instruções que o computador executa automaticamente. *Programar em Excel* refere-se ao fato de que é possível orientar o Excel a executar

automaticamente tarefas que em geral você faz manualmente – poupando muito do seu tempo (você espera) e reduzindo erros. Eu poderia continuar, mas preciso guardar alguma coisa boa para o Capítulo 1.

Se você leu até aqui, é seguro apostar que você precisa se tornar um programador de Excel. Isto pode ser algo que você decidiu por si mesmo, ou (mais provavelmente) algo que seu chefe impôs. Neste livro, eu lhe digo o suficiente sobre programação em Excel de modo que você não se sentirá um idiota na próxima vez que ficar preso em uma sala com um grupo de aficionados em Excel. E quando terminar este livro, você poderá dizer, honestamente, “Sim, eu programo em Excel”.

ExcelVBABokks.indb 2

15/02/2013 13:59:16

Introdução

3

Por Que se Preocupar?

A maioria dos usuários de Excel nunca se importa em explorar a programação VBA. O seu interesse neste tópico o coloca, definitivamente, em uma tropa de elite. Bem-vindo ao grupo! Se você ainda não está convencido de que se tornar um mestre em programação no Excel é uma boa ideia, tenho alguns bons motivos pelos quais você poderia querer dedicar algum tempo para aprender a programar VBA.

5 Isso o tornará mais competitivo no mercado de trabalho.

Gostando ou não, os aplicativos da Microsoft são extremamente populares. Você já deve saber que todas as aplicações da

Microsoft suportam VBA. Quanto mais você sabe sobre VBA,

melhores são as suas chances para progredir em seu trabalho.

5 Permitirá que você explore o máximo do seu investimento em software (ou, mais provavelmente o investimento do seu empregador). Usar Excel sem saber VBA é como comprar uma TV e assistir a apenas os canais ímpares.

5 Aumentará sua produtividade (eventualmente).

Definitivamente, demora algum tempo para conhecer VBA, mas esse tempo será compensado quando você for mais produtivo.

É como o que lhe disseram sobre ir para a faculdade.

5 É divertido (bem, algumas vezes). Algumas pessoas gostam, de fato, de fazer certas coisas com Excel que seriam impossíveis de outra forma. Quando você terminar este livro, poderá ser uma dessas pessoas.

Está convencido agora?

Penso que...

Normalmente, pessoas que escrevem livros têm em mente um leitor alvo. Neste livro, o meu leitor alvo é um conglomerado de dúzias de usuários de Excel que conheci através dos anos (tanto pessoalmente como no ciberespaço). Os seguintes pontos descrevem bem o meu hipotético leitor alvo:

5 Você tem acesso a um PC no trabalho – e provavelmente em casa.

5 Você usa Excel 2007 ou Excel 2010.

5 Você vem usando computadores há muitos anos.

5 Você usa Excel frequentemente no trabalho, e considera ser mais capaz no uso da ferramenta do que o público geral.

5 Você precisa que o Excel faça algumas coisas que atualmente não consegue que ele faça.

5 Você tem pouca ou nenhuma experiência em programação.

ExcelVBABokks.indb 3

15/02/2013 13:59:16

4

Programando Excel VBA Para Leigos, Tradução da 2a Edição

5 Você compreende que o sistema de Help do Excel pode

realmente ser útil. Encare os fatos, este livro não cobre tudo. Se você conseguir se entender com o sistema de Help, você

conseguirá achar as peças que faltam.

5 Você precisa concluir algum trabalho e possui pouca tolerância à livros grossos e chatos sobre computação.

Seção Obrigatória das

Convenções Tipográficas

Todos os livros de computação possuem uma seção sobre isto. Eu acho que existe alguma lei federal exigindo isso. Leia ou simplesmente pule esta etapa.

Algumas vezes, eu me refiro a combinações de teclas, o que significa que você pressionará uma tecla enquanto aperta outra, por exemplo, Ctrl + Z

significa que você pressiona a tecla Ctrl e a tecla Z ao mesmo tempo.

Nos comandos de menu, uso um caractere distinto para separar itens do menu. Por exemplo, para abrir um arquivo do livro-texto, você usa

o seguinte comando:

Arquivo⇒Abrir

Observe que no Excel 2007, não há algo como um menu “File” (Arquivo) visível em sua tela. No Excel 2007, o menu Arquivo foi substituído pelo Office Button (Botão Office), uma pequena engenhoca redonda que aparece do lado superior esquerdo de qualquer aplicativo do Office 2007, que implementou o que é chamado de Ribbon (Barra). A certa altura, a Microsoft resolveu que o Office Button não era uma ideia tão boa e o Excel 2010 descartou aquele Office Button redondo e o substituiu por uma guia de Botão chamada Arquivo. Neste livro, eu o chamo de “Arquivo”, portanto, se você usar Excel 2007, lembre-se apenas de que “Arquivo” significa “pequena engenhoca redonda do lado superior esquerdo”.

Qualquer texto que você precisar inserir aparece em **negrito**. Por exemplo, eu poderia dizer, entre com =**SUM(B:B)** na célula A1.

A programação em Excel envolve desenvolver *código* – isto é, as instruções que o Excel segue. Todo o código neste livro é apresentado com uma fonte como:

Range(“A1:A12”).Select

Alguns códigos de linhas longas não cabem nas margens deste livro. Em tais casos, eu uso a sequência de caracteres padrão VBA de continua-

ção de linha: um espaço seguido por um caractere de sublinhado. Eis um exemplo:

ExcelVBABokks.indb 4

15/02/2013 13:59:16

Introdução

5

Selection.PasteSpecial Paste: = xlValues, _

Operation: = xlNone, SkipBlanks: = False, _

Transpose: = False

Quando inserir este código, você poderá digitá-lo como está ou colocá-lo em uma linha única (retirando os espaços e os sublinhados).

Verifique Suas

Configurações de Segurança

Vivemos em um mundo cruel. Parece que há sempre um estrategista tentando obter vantagem de você ou causando algum tipo de problema.

O mundo da computação é igualmente cruel. Provavelmente, você sabe a respeito de vírus de computador, os quais podem causar coisas desagradáveis em seu sistema. Mas, você sabe que os vírus de computador também podem estar em um arquivo Excel? É verdade. De fato, é relativamente fácil escrever um vírus de computador usando VBA.

Inocentemente, um usuário pode abrir um arquivo Excel e espalhar o vírus para outros arquivos Excel, e para outros sistemas.

Com o passar dos anos, a Microsoft se tornou cada vez mais preocupada com problemas de segurança. Isto é uma coisa boa, mas também significa que usuários do Excel precisam entender como as coisas funcionam. Você pode checar as configurações de segurança do Excel clicando em Arquivo < Opções < Central de confiabilidade < Configurações da Central de Confiabilidade. Existe uma miríade de opções lá dentro, e corre o boato que nunca mais se ouviu falar das pessoas que abriram tal caixa de diálogo.

Se você clicar na guia Configurações de Macro (à esquerda da caixa de diálogo [Central de Confiabilidade]), as suas opções serão como a seguir:

5 Desabilitar todas as macros sem notificação: As macros não funcionarão, não importa o que você fizer.

5 Desabilitar todas as macros com notificação: Quando você abre um arquivo Excel com macros, você verá uma Barra de

Mensagem aberta com uma opção para você clicar e habilitar

as macros, ou (se o VBE estiver aberto), você receberá uma mensagem perguntando se quer habilitar as macros.

5 Desabilitar todas as macros, exceto as digitalmente assinadas:
Apenas macros com uma assinatura digital podem rodar (porém, até mesmo para aquelas assinaturas que não foram marcadas como confiáveis, você receberá o aviso de segurança).

5 Habilitar todas as macros (não recomendado; códigos possivelmente perigosos podem ser executados).

ExcelVBABokks.indb 5

15/02/2013 13:59:16

6

Programando Excel VBA Para Leigos, Tradução da 2a Edição

Imagine este cenário: Você passa uma semana escrevendo um programa VBA matador, que revolucionará a sua empresa. Você o testa cuidadosamente e depois envia ao seu chefe. Ele o chama ao seu escritório e reclama que a sua macro não faz absolutamente nada. O que está acontecendo? Possivelmente, as configurações de segurança do Excel de seu chefe não permitem a execução de macros. Ou talvez ele tenha decidido aceitar a sugestão padrão da Microsoft e desativar as macros ao abrir o arquivo.

A questão toda? Só porque uma planilha Excel contém uma macro, não garante que a macro será executada. Tudo depende da configuração de segurança e se o usuário decide ativar ou desativar macros para aquele arquivo.

Para trabalhar com este livro, será preciso habilitar as macros para os arquivos com os quais você trabalha. Meu conselho é usar o segundo nível de segurança. Então, quando abrir o arquivo que criou, você pode simplesmente habilitar as macros. Se você abrir um arquivo de alguém que não conhece, você deve desabilitar as macros e verificar o código VBA para ter certeza de que não possui nada destrutivo ou

malicioso.

Geralmente, é muito fácil identificar um código VBA suspeito.

Como Este Livro Está Organizado

Eu dividi este livro em seis partes importantes, cada qual contendo vários capítulos. Embora eu tenha arrumado os capítulos em uma sequência lógica, você pode lê-los em qualquer ordem que quiser. Eis uma rápida visão geral do que está guardado para você.

Parte I: Introdução ao VBA

A Parte I contém dois capítulos. No primeiro capítulo, apresento a linguagem VBA. No Capítulo 2, faço-o suar, levando-o para um passeio guiado.

Parte II: Como o VBA Funciona com Excel

Ao escrever este livro, eu assumi que você já sabe como usar Excel. Os quatro capítulos na Parte II mostram de forma mais clara como VBA é implementado no Excel. Todos esses capítulos são importantes, portanto, eu não recomendo que os pule, certo?

ExcelVBABokks.indb 6

15/02/2013 13:59:16

Introdução

7

Parte III: Conceitos de Programação

Os oito capítulos na Parte III o levam ao que realmente é a programação. Você talvez não precise saber todas essas coisas, mas ficará grato se estiverem lá quando você precisar.

Parte IV: Como se Comunicar com Seus Usuários

Uma das melhores partes de programar em Excel é desenvolver caixas de diálogo (bem, pelo menos, *eu* gosto). Os cinco capítulos da Parte IV lhe mostram como criar caixas de diálogo que parecem ter vindo

diretamente do laboratório de software da Microsoft.

Parte V: Juntando Tudo

Os dois capítulos da Parte V reúnem informações dos capítulos anteriores. Você descobrirá como incluir os seus próprios botões personalizados na interface de usuário do Excel e aprenderá como desenvolver funções personalizadas de planilhas, criar add-ins, projetar aplicativos orientados por usuário e até mesmo trabalhar com outros aplicativos Office.

Parte VI: A Parte dos Dez

Tradicionalmente, livros da série Para Leigos® possuem uma parte final que consiste de pequenos capítulos com listas práticas e informativas.

Porque eu sou um fã de tradições, este livro contém dois de tais capítulos, os quais você pode pesquisar quando de sua conveniência (se você for como a maioria dos leitores, irá para esta parte primeiro).

Espere, Há Mais!

Eu me entusiasmei e escrevi dois outros capítulos, que não caberiam neste livro, pois ultrapassei o limite de páginas. Então, eu os coloquei no site do livro, juntamente com os exemplos de arquivos (veja Como Obter os Arquivos de Exemplo mais adiante nesta Introdução). Os dois capítulos extras são Trabalhando com Cores e 10 Dicas Sobre o Que Fazer e o Que Não Fazer em VBA.

Ícones Usados Neste Livro

Em determinada ocasião, uma empresa de pesquisa de mercado deve ter mostrado aos editores que poderiam vender mais cópias de seus ExcelVBABokks.indb 7

15/02/2013 13:59:16

Programando Excel VBA Para Leigos, Tradução da 2a Edição

livros de computador se eles acrescentassem ícones nas margens daqueles livros. *Ícones* são aquelas pequenas figuras que supostamente chamam sua atenção para vários itens ou lhe ajudam a decidir se algo é digno de leitura.

Não sei se essa pesquisa é válida, mas não quero arriscar. Portanto, aqui estão os ícones que você encontrará em suas viagens da primeira à última capa.

Quando você vir este ícone, o código sendo discutido está disponível na Web. Faça o download dele para evitar muita digitação. Para mais informações, veja “Como Obter os Arquivos de Exemplos”.

Este ícone sinaliza material que pode ser considerado técnico. É

DE ESPECIALISTA

possível que você o julgue interessante, mas, se estiver com pressa, **O P A P**

pode pulá-lo.

DICA

Não pule informações marcadas com este ícone. Ele identifica um atalho que pode poupar muito do seu tempo (e talvez até permita que você saia do trabalho mais cedo).

BRE-SE

Este ícone diz quando você precisa armazenar informações nos **recessos** de seu cérebro para uso posterior.

Leia tudo o que estiver marcado com este ícone. Caso contrário, você **CUIDADO!**

pode perder seus dados, explodir seu computador, causar uma fusão nuclear – ou talvez até arruinar todo o seu dia.

Como Obter os Arquivos de Exemplos

Visite o site da Alta Books para acessar a página deste livro e fazer o download dos arquivos de exemplos, bem como ver os Capítulos Extras (Bônus): www.altabooks.com.br (procure pelo nome do livro).

Com os arquivos de exemplos, você poupará muita digitação. Melhor ainda, será possível brincar com eles e tentar diversas alterações. Na verdade, eu recomendo enfaticamente que você brinque com esses

arquivos. A melhor maneira de dominar VBA é experimentando.

ExcelVBABokks.indb 8

15/02/2013 13:59:17

Introdução

9

E Agora?

Ler esta introdução foi o primeiro passo. Agora, é hora de ir em frente e se tornar um programador.

Se você for um programador iniciante, sugiro que inicie pelo Capítulo 1

e siga o livro até ter descoberto o suficiente para fazer o que deseja. O

Capítulo 2 oferece alguma experiência prática imediata, assim você terá a ilusão de que está progredindo.

Mas este é um país livre (pelo menos na hora que escrevia estas palavras). Por isso, eu não denunciarei você caso resolva consultá-lo aleatoriamente e ler o que aguçar o seu gosto.

Espero que você se divirta muito lendo este livro, tanto quanto eu o fiz ao escrevê-lo.

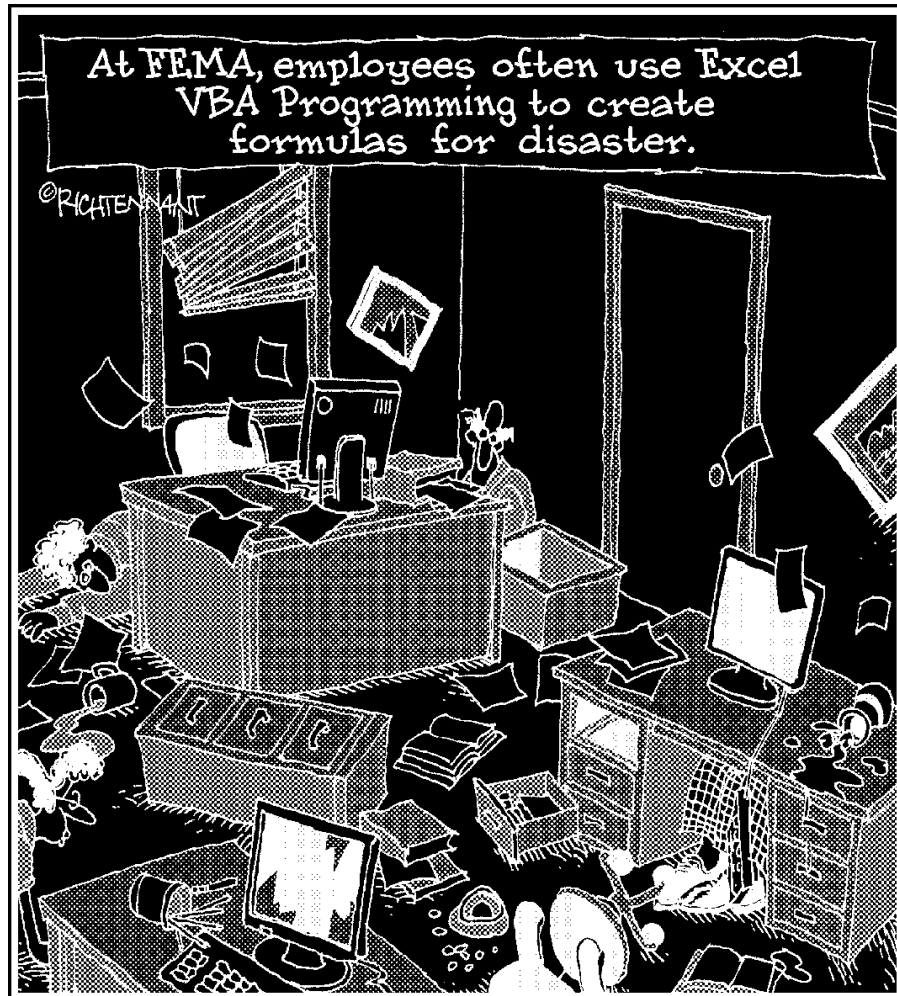
ExcelVBABokks.indb 9

15/02/2013 13:59:17

10 Programando Excel VBA Para Leigos, Tradução da 2a Edição

ExcelVBABokks.indb 10

15/02/2013 13:59:17



Parte I

Introdução ao VBA

A 5ª Onda

Por Rich Tennant

Na ARCA*, geralmente os funcionários usam Excel com
Programação VBA para criar fórmulas para desastre.

eio

aos Alh

a de C

egulador

cia R

* Agên

ExcelVBABokks.indb 11

15/02/2013 13:59:18

Nesta parte...

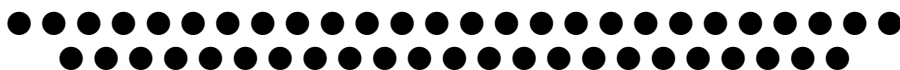
Cada livro precisa começar em algum lugar. Este começa apresentando-o ao Visual Basic para Aplicativos (e tenho certeza que vocês dois se tornarão grandes amigos no decorrer de algumas dúzias de capítulos). Depois de feitas as apresentações, o Capítulo 2 o levará através de uma seção de programação da vida real do Excel.

ExcelVBABokks.indb 12

15/02/2013 13:59:18

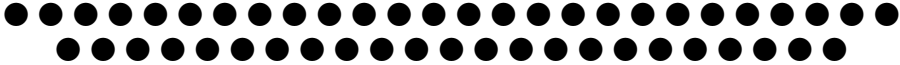
Capítulo 1

O Que É VBA ?



Neste Capítulo

- ☐ Como conseguir uma visão geral dos conceitos do VBA
- ☐ Descubra o que se pode fazer com o VBA
- ☐ Descubra as vantagens e desvantagens de usar o VBA
- ☐ Uma pequena aula de história sobre o Excel



S e você está ansioso para pular na programação de VBA, segure-se um pouco. Este capítulo é totalmente desprovido de qualquer material de treinamento prático. No entanto, ele contém algumas informa-

ções essenciais de apoio que o ajudam a se tornar um programador de Excel. Em outras palavras, este capítulo prepara o caminho para tudo que vem pela frente e dá a você um sentido de como a programação de Excel se ajusta no esquema geral do universo. Não aborrece tanto como você poderia imaginar.

Tudo Bem, Então o Que é VBA?

VBA, que significa *Visual Basic for Applications* (Visual Basic para aplicativos), é uma linguagem de programação desenvolvida pela Microsoft – você sabe, a empresa que tenta fazê-lo comprar uma nova versão do Windows a cada ano. Excel, juntamente com outros membros do Microsoft Office, inclui a linguagem VBA (sem custos extras).

Resumidamente, VBA é uma ferramenta que pessoas como você e eu usamos para desenvolver programas que controlam o Excel.

Imagine um robô inteligente que saiba tudo sobre o Excel. Esse robô pode ler instruções e pode também operar o Excel com muita rapidez e precisão. Quando você quiser que o robô faça algo no Excel, você programa as instruções dele, usando códigos especiais. Diga ao robô para seguir suas instruções, enquanto você se senta e relaxa, tomando um copo de limonada. Isso que é VBA, uma linguagem de código para os robôs. Mas, veja, o Excel não vem com um robô e nem faz limonada.

ExcelVBABokks.indb 13

15/02/2013 13:59:18

14 Parte I: Introdução ao VBA

DE ESPECIALISTA

O P A P

Algumas palavras sobre terminologia

A terminologia de programação em Excel plo, se você escrever uma macro que acres-pode ser um pouco confusa. Por exemplo, centa cor a algumas células, imprime a pla-VBA é uma linguagem de programação, nilha e depois remove a cor, essas três mas também serve como uma linguagem etapas foram automatizadas.

de macro. Como você domina algo escrito

em VBA e executado em Excel? É uma ma—

A propósito, macro não é um acrônimo de

cro ou um programa? Normalmente, o sis-

Messy And Confusing Repeated Operation

tema Help (Ajuda) do Excel se refere aos (Operação Repetida Confusa e Desordena-procedimentos VBA como macros, assim, da). Ao invés disso, ela vem da palavra gre-eu uso essa terminologia.

ga makros, que significa grande – que também descreve o seu contracheque depois

Uso o termo *automatizar* neste livro. Esse de se tornar um programador especialista termo significa que uma série de etapas são em macro.

completadas automaticamente. Por exem-

O Que Você Pode Fazer com VBA?

Você, provavelmente, está ciente que os usuários do Excel usam o programa para milhares de diferentes tarefas. Seguem alguns exemplos: 5 Análise de dados científicos

5 Preparação de orçamentos e previsões financeiras

5 Criação de faturas e outros formulários

5 Desenvolvimento de gráficos de dados

5 Manutenção de listagens de assuntos como nomes dos clientes, notas dos alunos ou ideias para presentes (um lindo bolo de frutas seria adorável)

5 Etc., etc., etc.

Os exemplos são muitos, mas acredito que você já entendeu. O que quero dizer é que o Excel é usado para coisas variadas, e qualquer um que ler este livro tem diferentes necessidades e expectativas consideráveis. Uma coisa que todos os leitores têm em comum é a necessidade de automatizar algum aspecto do Excel. Isso, meu caro leitor, é o que VBA significa.

Por exemplo, seria possível criar um programa VBA para importar alguns números e depois, formatar e imprimir o seu relatório de vendas ao final do mês. Depois de desenvolver e testar o programa, você pode executar a macro com um único comando, levando o Excel a executar automaticamente muitos procedimentos demorados. Ao invés de lutar através de uma cansativa sequência de comandos, você pode pegar uma caneca de suco e deixar o seu computador fazer o trabalho – exatamente o que ele deveria fazer, certo?

ExcelVBABokks.indb 14

15/02/2013 13:59:18

Capítulo 1: O Que é VBA ?

15

Nas seções seguintes, descreverei, em poucas palavras, alguns usos comuns para as macros VBA. Um ou dois destes fará que você se ligue.

Inserir um grupo de texto

Se com frequência você precisa entrar com o nome da empresa, endereço e número de telefone em sua planilha de trabalho, é possível criar uma macro para digitar essas informações, desde que seja o que você quer. Por exemplo, você pode desenvolver uma macro que digite

automaticamente uma lista de todas as pessoas de vendas que trabalham para a sua empresa.

Automatizar tarefas executadas com frequência

Suponhamos que você é o gerente de vendas e precisa preparar o relatório de vendas do fim do mês para satisfazer seu chefe. Se for uma tarefa direta, você pode desenvolver um programa VBA para executá-la.

O seu chefe ficará impressionado com a qualidade e consistência dos seus relatórios e você pode ser promovido a um novo cargo para o qual você está altamente desqualificado.

Automatizar operações repetitivas

Se você precisa executar 12 vezes uma mesma ação em, digamos, 12

diferentes planilhas do Excel, enquanto executa a primeira planilha, você pode gravar uma macro e deixar que a macro repita a sua ação nas outras planilhas. O melhor disso é que o Excel nunca reclama que está aborrecido. A gravação de uma macro no Excel é similar à grava-

ção de uma ação ao vivo com um gravador de vídeo. A diferença é que ele não requer uma câmera e a bateria nunca precisa ser recarregada.

Criar um comando personalizado

Geralmente, você usa a mesma sequência de comandos no Excel? Se afirmativo, é possível poupar alguns segundos, desenvolvendo uma macro que combine esses comandos em um único comando personalizado, que você executa com um único toque de teclado ou clique de botão.

Provavelmente, você não poupa tanto tempo, mas certamente será mais exato. E o cara do cubículo ao lado ficará realmente impressionado.

ExcelVBABokks.indb 15

15/02/2013 13:59:18

16 Parte I: Introdução ao VBA

Criar um botão personalizado

Você pode personalizar sua barra de ferramentas Página Inicial com os seus próprios botões, que executam as macros escritas por você. Os outros funcionários do escritório costumam ficar muito impressionados com botões que fazem mágica.

Se você estiver usando o Excel 2010, pode adicionar botões à barra superior, algo que não é possível no Excel 2007.

Desenvolver novas funções de planilhas

Embora o Excel inclua diversas funções (tais como SOMA e MÉDIA), é possível criar funções de planilha de trabalho personalizadas, que podem simplificar muito as suas fórmulas. Eu garanto que você ficará surpreso com quão fácil é. Mostro como fazer isso no Capítulo 20. E

melhor ainda, a caixa de diálogo Insert Function (Inserir Função) exibe as suas funções personalizadas, fazendo com que elas pareçam internas. Coisa muito boa.

Criar aplicativos completos,

guiados por macro

Se você estiver disposto a gastar algum tempo, pode usar VBA para aplicativos completos em larga escala, com uma guia Ribbon, caixas de diálogo, ajuda na tela e muitas outras benesses. Este livro não vai tão longe, mas estou falando sobre isso para incentivá-lo quanto a quão poderoso o VBA é realmente.

Criar suplementos (add-ins)

personalizados para o Excel

Você provavelmente está familiarizado com os suplementos (add-ins) do Excel. Por exemplo, Analysis ToolPak (Pacote de Ferramentas de Análise) é um suplemento popular. Você pode usar o VBA para desenvolver os seus próprios add-ins de objetivo especial. Desenvolvi a minha Caixa de Utilidades usando apenas VBA e as pessoas do mundo todo me pagam um bom dinheiro para poderem usá-la.

Vantagens e Desvantagens do VBA

Nesta parte, descrevo resumidamente as boas coisas sobre VBA — e também exploro o seu lado mais escuro.

Capítulo 1: O Que é VBA ?

17

Vantagens do VBA

É possível automatizar quase tudo o que você faz em Excel. Para fazê-lo, basta escrever instruções que o Excel as executa. Automatizar uma tarefa usando VBA oferece várias vantagens:

5 O Excel sempre executa tarefas exatamente do mesmo jeito (na maioria dos casos, consistência é uma coisa boa).

5 O Excel executa a tarefa muito mais depressa do que você pode fazer manualmente (a menos, é claro, que você seja o Super-Homem).

5 Se você for um bom programador de macros, o Excel sempre executa tarefas sem erros (já não podemos dizer isso sobre eu e você).

5 Se você configurar as coisas corretamente, qualquer um sem conhecimento do Excel pode executar a tarefa.

5 É possível fazer coisas em Excel que poderiam parecer impossíveis — o que pode torná-lo uma pessoa muito popular no escritório.

5 Para tarefas longas e demoradas, você não precisa ficar sentado diante de seu computador e se aborrecer. O Excel faz o trabalho, enquanto você fica à toa.

É justo que eu use o mesmo tempo para relacionar as desvantagens (ou possíveis desvantagens) do VBA:

5 Você tem que saber como escrever programas em VBA (mas foi para isso que você comprou este livro, certo?). Felizmente, não é tão complicado quanto você poderia prever.

5 Outras pessoas que precisem usar os seus programas VBA devem ter suas próprias cópias do Excel. Seria muito bom se você pressionasse um botão e o aplicativo Excel/VBA se convertesse em um programa autônomo, mas isso não é possível (e provavelmente nunca será).

5 Às vezes, as coisas saem erradas. Em outras palavras, não é possível supor, cegamente, que o seu programa VBA sempre funcionará corretamente em todas as circunstâncias. Bem-vindo ao mundo da depuração e, se outros estiverem usando as suas macros, ao suporte técnico.

5 VBA é como um alvo em movimento. Como você sabe, a Microsoft continuamente atualiza o Excel. Mesmo a Microsoft se esforçando para que haja uma compatibilidade entre as versões, você pode descobrir que o código VBA que escreveu não funciona adequadamente com versões mais antigas ou com uma versão futura de Excel.

18 Parte I: Introdução ao VBA

VBA Resumidamente

Só para saber onde você está se metendo, preparei um rápido resumo sobre VBA. E claro, descrevi também todas as coisas dolorosas que detalharei neste livro.

5 Ações são executadas em VBA escrevendo (ou gravando) código em um módulo VBA. Você vê e edita módulos VBA, usando o Visual Basic Editor (VBE).

5 Um módulo VBA consiste de Sub procedimentos (secundários).

Um procedimento secundário nada tem a ver com submarinos ou

sanduíches saborosos. Ao contrário, trata-se de um grupo de código de computador que realiza alguma ação em ou com objetos (a ser discutido em breve). O exemplo a seguir mostra um simples procedimento de Sub, chamado AddEmUp. Esse incrível programa

exibe o resultado de 1 mais 1.

```
Sub AddEmUp()
```

```
Sum = 1 + 1
```

```
MsgBox "The answer is " & Sum
```

```
End Sub
```

Um procedimento Sub que não executa adequadamente é chamado de substandard (um secundário padrão).

5 Um módulo VBA também pode ter procedimentos de Function.

Um procedimento de Function retorna um único valor. É possível chamá-lo a partir de outro procedimento VBA ou até usá-lo como uma função em uma fórmula de planilha. A seguir, está um exemplo de um

procedimento Function (chamado de AddTwo). Essa Function aceita dois números (chamados de argumentos) e retorna a soma daqueles valores.

```
Function AddTwo(arg1, arg2)
```

```
AddTwo = arg1 + arg2
```

```
End Function
```

Um procedimento de Function que não funciona corretamente é dito disfuncional.

5 Objetos manipulados VBA. O Excel oferece dúzias e dúzias de objetos que podem ser manipulados. Exemplos de objetos incluem um livro de registro, uma planilha, uma faixa de célula, um gráfico e uma pasta de trabalho. Há muitos outros objetos à sua disposição, e você pode manipulá-los usando código VBA.

5 Objetos VBA são organizados em uma hierarquia. Os objetos podem agir como contêineres para outros objetos. O Excel está no alto da hierarquia de objetos. O próprio Excel é um objeto chamado Application (Aplicativo). O objeto Application contém outros objetos, tais como objetos Workbook e objetos Add-Ins. O objeto Workbook pode conter outros objetos, tais como objetos Worksheet e objetos Chart. Um objeto Worksheet pode conter objetos como ExcelVBABokks.indb 18

15/02/2013 13:59:18

Capítulo 1: O Que é VBA ?

19

objeto Range e objeto PivotTable. O termo modelo de objeto refere-se à organização desses objetos (mais detalhes sobre modelo de objeto podem ser encontrados no Capítulo 4).

5 Objetos do mesmo tipo formam uma coleção. Por exemplo, a coleção Worksheets consiste de todas as planilhas de uma pasta de trabalho específico. A coleção Charts consiste de todos os objetos Chart de uma pasta de trabalho. As próprias coleções são objetos.

5 Você se refere a um objeto, especificando a sua posição na hierarquia de objeto, usando um ponto como um separador. Por exemplo, é possível fazer referência ao livro de trabalho Book1.xlsx, assim: Application.Workbooks(“Book1.xlsx”) Isso se refere à pasta de trabalho Book1.xlsx na coleção Workbooks.

Essa coleção está contida no objeto Application (ou seja, Excel).

Levando isso a outro nível, você pode se referir a Sheet1 em Book1.xlsx Application.Workbooks(“Book1.xlsx”). _

Worksheets(“Sheet1”)

Conforme mostrado no exemplo a seguir, ainda é possível levar isso a um outro nível e fazer referência a uma célula específica (neste caso, célula A1):

Application.Workbooks(“Book1.xlsx”).

Worksheets(“Sheet1”).Range(“A1”)

5 Se você omitir referências específicas, o Excel usa objetos ativos.

Se Book1.xlsx for a pasta de trabalho ativa, é possível simplificar a referência anterior como a seguir:

Worksheets(“Sheet1”).Range(“A1”)

Se você souber que Sheet1 é a planilha ativa, pode simplificar ainda mais a referência:

Range(“A1”)

5 Os objetos têm propriedades. É possível pensar sobre uma propriedade como a configuração de um objeto. Por exemplo, um objeto Range tem propriedades como Value e Address. Um objeto Chart tem propriedades como HasTitle e Type. Você pode usar VBA

para determinar propriedades do objeto e também alterar suas propriedades.

5 Uma propriedade é referenciada a um objeto combinando o

nome do objeto com o nome da propriedade, separados por um ponto. Por exemplo, você pode referenciar a propriedade Value na célula A1 em Sheet1 assim:

```
Worksheets("Sheet1").Range("A1").Value
```

ExcelVBABokks.indb 19

15/02/2013 13:59:18

20 Parte I: Introdução ao VBA

5 Você pode atribuir valores a variáveis. Uma variável é um elemento nomeado que armazena informações. É possível usar

variáveis em seu código VBA para armazenar coisas como

valores, texto ou configurações de propriedade. Para atribuir um valor na célula A1 em Sheet1 a uma variável chamada

Interest, use a seguinte declaração VBA:

```
Interest = Worksheets("Sheet1").Range("A1"). Value
```

5 Objetos têm métodos. Um método é uma ação que Excel executa com um objeto. Por exemplo, um dos métodos em um

objeto Range é ClearContents. Esse método nomeado

adequadamente, limpa o conteúdo de uma faixa de células.

5 Você especifica um método combinando o objeto com o método, separados por um ponto. Por exemplo, a seguinte

declaração limpa o conteúdo da célula A1:

```
Worksheets("Sheet1").Range("A1").ClearContents
```

5 VBA inclui todas as construções de linguagens modernas de programação, inclusive arrays e loops. Em outras palavras, se você estiver disposto a gastar um pouco de tempo administrando o assunto, pode escrever códigos que fazem

algumas coisas incríveis.

Acredite ou não, a lista anterior descreve bem o VBA resumidamente.

Agora, você só precisa encontrar os detalhes. Por isso é que este livro tem mais páginas.

Uma Excursão pelas Versões Anteriores

Se você pretende desenvolver macros VBA, deve ter algum conhecimento sobre a história do Excel. Eu sei que você não estava esperando ter uma aula de história ao escolher este livro, mas acredite em mim, isso é uma coisa importante que pode levá-lo a fazer sucesso na próxima festa de nerds.

Aqui estão todas as principais versões de Excel que surgiram para Windows, bem como algumas palavras sobre como elas lidam com macros:

5 Excel 2: A versão original de Excel para Windows foi chamada de Versão 2 (ao invés de 1), para que ela pudesse ser

correspondente à versão Macintosh. O Excel 2 apareceu pela primeira vez em 1987, porém ninguém a usa mais, portanto, você pode muito bem esquecer que ela existiu.

5 Excel 3: Lançada em fins de 1990, essa versão apresenta a linguagem macro XLM. Também, ninguém mais usa essa versão.

Capítulo 1: O Que é VBA ?

21

5 Excel 4: Essa versão chegou às ruas no início de 1992. Ela também usa a linguagem macro XLM. Uma pequena quantidade de pessoas ainda usa essa versão (elas são associadas à filosofia, *se ainda não quebrou, por que trocar?*).

5 Excel 5: Esta surgiu no início de 1994. Foi a primeira versão a usar VBA (mas também suporta XLM). Usuários de Excel 5 estão se tornando incrivelmente raros).

5 Excel 95: Tecnicamente conhecida como Excel 7 (não há Excel 6), essa versão começou a surgir no verão de 1995. Ela é uma versão de 32-bits e exige Windows 95 ou Windows NT. Ela tem alguns aperfeiçoamentos VBA e suporta a linguagem XLM.

5 Excel 97: Esta versão (também conhecida como Excel 8), nasceu em Janeiro de 1997. Ela tem muitos aperfeiçoamentos e apresenta uma interface totalmente nova para programar macros VBA. O Excel 97 também usa um novo formato de arquivo (que versões anteriores de Excel não podiam abrir). Eventualmente, eu me deparei com alguém que ainda usa essa versão.

5 Excel 2000: O esquema de numeração dessa versão pulou para quatro dígitos. O Excel 2000 (também conhecido como Excel 9), fez

sua primeira aparição em junho de 1999. Ela inclui apenas alguns aperfeiçoamentos sob a perspectiva de um programador, com a maioria deles sendo para usuário – especialmente,

usuários online. Com o Excel 2000, veio a opção de assinar

macros digitalmente, permitindo assim a você a garantia de que o código entregue aos seus usuários são seus, de fato. O Excel 2000 ainda tem uma quantidade modesta de usuários.

5 Excel 2002: Essa versão (também conhecida como Excel 10 ou Excel XP) apareceu ao final de 2001. Talvez o recurso mais

significativo dessa versão seja a habilidade de recuperar o seu trabalho quando o Excel falha. Essa é também a primeira versão a usar proteção de cópia (conhecido como *product activation* ativação de produto).

5 Excel 2003: De todas as atualizações do Excel que eu já vi (e as vi todas), o Excel 2003 tem a menor quantidade de recursos

novos. Em outras palavras, a maioria dos usuários cativos do

Excel ficaram muito desapontados com o Excel 2003. Ainda

assim, as pessoas o compravam. Creio que foram esses

camaradas que vieram de uma versão pré Excel 2002. Enquanto

escrevo isto, provavelmente o Excel 2003 é a versão mais usada.

5 Excel 2007: O Excel 2007 marcou o início de uma nova era.

Excel 2007 se desfez do antigo menu e da barra de ferramentas da interface e apresentou a Faixa de Opções. Ele também

possibilita planilhas de trabalho muito maiores — mais de um milhão de linhas.

5 Excel 2010: O mais recente e, sem dúvida, o melhor. A Microsoft se superou com esta versão. Esta versão tem alguns novos recursos engenhosos (tais como gráficos sparkline), e também executa um pouco melhor em alguns aspectos. E se você precisar realmente de pastas de trabalho enormes, pode instalar a versão de 64-bits.

ExcelVBABokks.indb 21

15/02/2013 13:59:18

22 Parte I: Introdução ao VBA

Este livro é escrito para Excel 2007 e Excel 2010, portanto, se você não **BRE-SE**

tiver uma dessas versões, corre o risco de ficar confuso em alguns casos.

LEM

Então, qual é a finalidade dessa pequena aula de história? Se você pretende distribuir seus arquivos Excel/VBA a outros usuários, é de suma importância entender qual versão do Excel eles usam. Pessoas usando uma versão mais antiga podem não ser capazes de usufruir a vantagem de recursos introduzidos nas versões mais recentes. Por exemplo, se você escrever código VBA que faz referências à célula XFD1048576 (a última célula em uma planilha) aqueles que usam uma versão anterior ao Excel 2007 receberão um erro, pois versões anteriores ao Excel 2007 só tinham 65.536 linhas e 255 colunas (a última célula é IV65536).

O Excel 2007 e o Excel 2010 também têm alguns novos objetos, métodos e propriedades. Se você os usa em seu código, usuários com uma versão mais antiga do Excel receberão um erro quando rodarem sua macro — e você será responsabilizado. No entanto, a Microsoft disponibilizou um Pacote Office de Compatibilidade, o qual permite aos usuários de Excel 2003 e Excel XP abrir e salvar pastas de trabalho no novo formato de arquivo. Esse produto (que, a propósito, é gratuito) não tem os novos recursos dessas versões. Ele simplesmente permite que eles abram e salvem arquivos no formato de arquivo Excel 2007/2010.

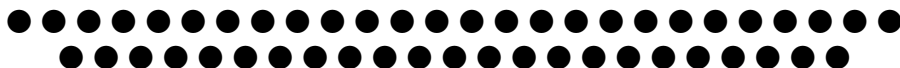
ExcelVBABokks.indb 22

15/02/2013 13:59:18

Capítulo 2

Saltando Para o

Lugar Certo



Neste Capítulo

- ☐ Como desenvolver uma macro VBA útil? Um exemplo prático, passo a passo
- ☐ Grave suas ações com o gravador de macros do Excel
- ☐ Como examinar e testar o código gravado
- ☐ Como lidar com questões de segurança de macro
- ☐ Mude uma macro gravada



E u não sou bem um nadador, mas descobri que a melhor maneira de um corpo frio entrar na água é pular direto — não faz sentido prolongar a agonia. Percorrendo este capítulo, você molha os pés imediatamente, mas evita mergulhar de cabeça.

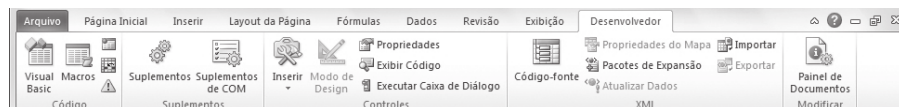
Quando chegar ao final deste capítulo, você pode começar a se sentir melhor quanto à questão dessa programação em Excel, e ficará satisfeito de ter dado o mergulho. Este capítulo oferece uma demonstração passo a passo de como desenvolver uma macro VBA simples, mas útil.

Primeiros Passos

Antes de se autoproclamar programador do Excel, você deve passar pelos rituais de iniciação. Isso significa que você precisa fazer uma pequena mudança, para que o Excel exiba uma nova guia no alto da tela: Desenvolvedor.

Ao clicar na guia Desenvolvedor, são exibidas informações do interesse dos programadores (este é você!). A Figura 2-1 mostra como a Faixa de Opções se parece quando a guia Desenvolvedor é selecionada.

15/02/2013 13:59:18



24 Parte I: Introdução ao VBA

Figura 2-1:

Normalmente, a guia

Desenvolvedor está

oculta, mas

é fácil

exibi-la.

A guia Desenvolvedor não é visível quando você abre o Excel, preciso dizer a ele para mostrá-la. Fazer com que Excel exiba a guia Desenvolvedor é fácil (e você só precisa fazê-lo uma vez). Mas, o procedimento varia, dependendo de qual versão você usa.

Usuários de Excel 2010

Siga estas etapas:

1. **Clique com o botão direito em qualquer parte da Faixa de Opções e escolha Personalizar a Faixa de Opções.**
2. **Na guia Personalizar a Faixa de Opções da caixa de diálogo Opções do Excel, localize Desenvolvedor na segunda coluna.**
3. **Coloque uma marca de verificação em Desenvolvedor.**
4. **Clique OK e você volta para o Excel com uma guia nova: Desenvolvedor.**

Usuários de Excel 2007

Siga estas etapas:

1. **Escolha Arquivo⇒Opções do Excel.**

Lembre-se de que no Excel 2007, o comando Arquivo significa clicar o botão redondo no canto superior esquerdo.

2. Na caixa de diálogo Opções, selecione Personalizar.

3. Coloque uma marca de verificação na caixa de seleção Mostrar guia Desenvolvedor na Faixa de Opções.

4. Clique OK para ver a nova guia Desenvolvedor ser exibida na Faixa de Opções.

ExcelVBABokks.indb 24

15/02/2013 13:59:18

Capítulo 2: Saltando Para o Lugar Certo

25

O Que Você Fará

Nesta seção, descrevo como criar a sua primeira macro. A macro que você está prestes a criar fará isto:

5 Digite seu nome em uma célula.

5 Forneça a data e hora atuais na célula abaixo.

5 Formate as duas células para exibir em negrito.

5 Mude o tamanho da fonte de ambas as células para 16 pontos.

Esta macro não ganhará quaisquer prêmios na Competição Anual dos Programadores de VBA, mas todas as pessoas têm que começar em algum lugar. A macro executa todas essas etapas em uma única ação.

Como descrevo nas próximas seções, você começa gravando as suas

ações na medida em que percorre essas etapas. Depois, você testa a macro, para ver se ela funciona. Por fim, você edita a macro para acrescentar alguns toques finais. Pronto?

Dando os Primeiros Passos

Esta seção descreve as etapas que você percorre antes de gravar a macro. Em outras palavras, você precisa de alguns preparativos antes do bom da festa começar.

- 1. Inicie o Excel, se ele ainda não estiver sendo executado.**
- 2. Se for necessário, crie uma nova planilha (Ctrl + N é a minha maneira preferida de fazer isso).**
- 3. Clique a guia Desenvolvedor e dê uma olhada no botão (Usar Referências Relativas), no grupo Código.**

Se a cor desse botão for diferente da dos outros, quer dizer que você está bem. Se o botão Usar Referências Relativas for da

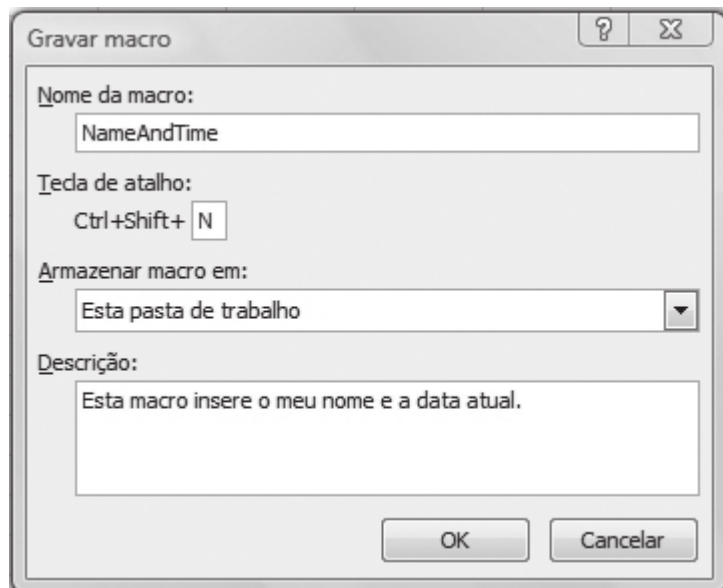
mesma cor dos outros botões, então será preciso clicá-lo.

Explicarei mais sobre o botão Usar Referências Relativas no Capítulo 6.

Por ora, vamos garantir que a opção esteja ativada. Quando ela estiver ativada, terá uma cor diferente.

ExcelVBABokks.indb 25

15/02/2013 13:59:18



26 Parte I: Introdução ao VBA

Gravando a Macro

Aqui está a parte prática. Siga as seguintes instruções cuidadosamente:

- 1. Selecione uma célula — qualquer célula que queira**
- 2. Escolha Desenvolvedor⇒Código⇒Gravar Macro ou clique o botão de gravação de macro na barra de status.**

A caixa de diálogo Gravar Macro aparece, conforme mostrado na Figura 2-2.

Figura 2-2:

A caixa de

diálogo

Gravar

Macro

aparece

quando

você for

gravar

uma macro

3. Entre com um nome para a macro.

O Excel oferece um nome padrão, mas é melhor usar um nome mais descritivo. NameAndTime (sem espaços) é um bom nome para essa macro.

4. Clique na caixa Tecla de Atalho e entre com Shift + N (para um N

em maiúscula), como a tecla de atalho.

Especificar uma tecla de atalho é opcional. Se você especificar um, então pode executar a macro pressionando a combinação, no caso, Ctrl + Shift + N.

5. Assegure-se de que a configuração de Armazenar macro seja esta pasta de trabalho.

6. Se quiser, você pode entrar com algum texto na caixa Descrição.

Isso é opcional. Algumas pessoas gostam de descrever o que a macro faz (ou o que supostamente faz).

7. Clique OK.

A caixa de diálogo fecha e o gravador de macro do Excel é ativado.

A partir deste ponto, o Excel monitora tudo o que você faz e converte para código VBA.

8. Digite seu nome em uma célula ativa.

9. Mova o indicador da célula para a célula abaixo e entre com a fórmula:

=AGORA()

A fórmula exibe a data e hora atuais.

15/02/2013 13:59:18

Capítulo 2: Saltando Para o Lugar Certo

27

10. Selecione a fórmula da célula e pressione Ctrl + C para copiar a célula na Área de Transferência.

11. Escolha Página Inicial⇒Área de Transferência⇒Colar⇒Colar Valores.

Este comando converte a fórmula aos seus valores.

12. Com a célula de data selecionada, pressione Shift + up para selecionar aquela célula e uma acima dele (a qual contém o seu nome).

13. Use os controles no grupo Página Inicial⇒Fonte para mudar a formatação para Negrito e mudar o tamanho da fonte para 16 pontos.

14. Escolha Desenvolvedor⇒Código⇒Parar Gravação.

O gravador de macro é desativado.

Parabéns! Você acabou de criar a primeira macro VBA no Excel. Talvez queira ligar para sua mãe e amigos e contar a boa notícia.

Testando a Macro

Agora, você pode testar a macro e ver se ela funciona corretamente.

Para testar sua macro, mova para uma célula vazia e pressione Ctrl + Shift + N.

Em um piscar de olhos, o Excel executará a macro. Seu nome e a data atual aparecerão em negrito e em letras grandes.

Outra maneira de executar a macro é selecionar

DICA

Desenvolvedor⇒Código⇒Macros (ou pressionar Alt + F8) para exibir a caixa de diálogo Macros. Selecione a macro da lista (neste caso, NameAndTime) e clique em Executar. Assegure-se de selecionar a célula que contenha o seu nome antes de executar a macro.

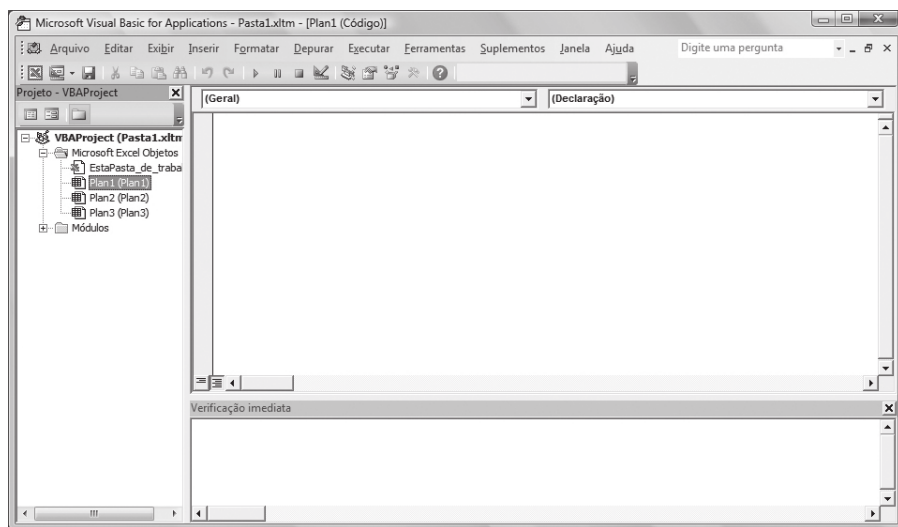
Examinando a Macro

Até agora, você gravou uma macro e a testou. Se for do tipo curioso, provavelmente você está imaginando como essa macro se parece. E pode até pensar onde ela está armazenada.

Lembra quando você começou a gravar a macro? Você indicou que o Excel deveria armazenar a macro na Pasta de Trabalho. A macro é armazenada na planilha, mas você precisa ativar o Visual Basic Editor (VBE) para vê-la.

ExcelVBABokks.indb 27

15/02/2013 13:59:18



28 Parte I: Introdução ao VBA

Siga os seguintes passos para ver a macro:

1. Escolha Desenvolvedor⇒Código⇒Visual Basic (ou pressione Alt + F11).

A janela do programa Visual Basic Editor aparece, conforme

mostrado na Figura 2-3. Essa janela é altamente personalizável, portanto, a sua janela VBE pode parecer um pouco diferente. A janela do programa VBE contém várias outras janelas e isso, provavelmente, é muito intimidador. Não se aflija: você se acostumará com isso.

2. Na janela VBE, localize a janela chamada de Projeto.

A janela Projeto (também conhecida como janela Project Explorer) contém uma lista de todas as planilhas e add-ins que estão abertas no momento. Cada projeto é colocado como uma árvore e pode ser expandido (para mostrar mais informações) ou ser contraído (para mostrar menos informações).

Figura 2-3: É

no Visual

Basic Editor

que você vê

e edita o

código VBA.

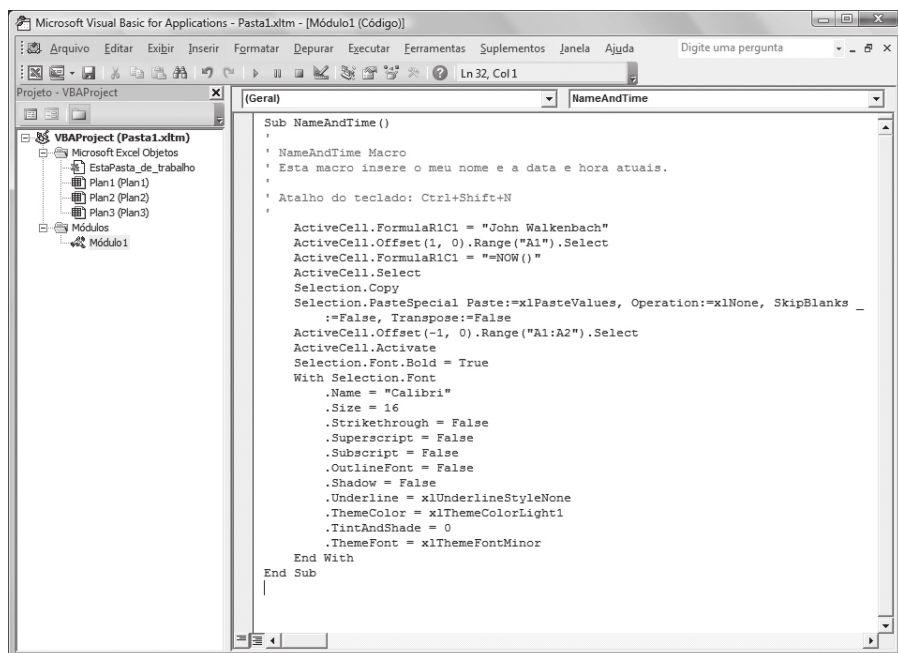
O VBE usa algumas janelas bem diferentes, qualquer delas podem-
DICA

do ser aberta ou fechada. Se uma janela não estiver imediatamente visível no VBE, você pode escolher uma opção a partir do menu Exibir para exibi-la. Por exemplo, se a janela Projeto não estiver visível, é possível escolher Exibir⇒Project Explorer (ou pressionar Ctrl + R) para exibi-la. Você pode exibir qualquer outra janela do VBE da mesma forma. Explico mais sobre os componentes do Visual Basic Editor no Capítulo 3.

3. Selecione o projeto que corresponde à planilha onde você gravou a macro.

Se você não salvou a planilha, o projeto provavelmente está com o nome VBAProject (pasta1).

4. Clique no sinal de adição (+) à esquerda da pasta chamada Módulos.



Capítulo 2: Saltando Para o Lugar Certo

29

A árvore se expande para mostrar Módulo1, que é o único módulo no projeto.

5. Clique duas vezes em Módulo1.

O código VBA nesse módulo é exibido na janela Código. A Figu—

ra2-4 mostra como ela se parece na minha tela. A sua tela pode não ser exatamente igual.

Figura 2-4: O

VBE mostra

o código

VBA no

Módulo1 da

planilha.

Nessa altura, provavelmente a macro parece com grego para você. Não se preocupe. Viaje por alguns capítulos e tudo ficará tão claro como a vista a partir do Olimpo.

A macro NameAndTime (também conhecida como uma Sub procedure) consiste de várias declarações. O Excel executa as declarações uma por uma, de cima para baixo. Uma declaração precedida por um apóstrofo (') é um comentário. Comentários estão incluídos apenas para sua informação e são essencialmente ignorados. Em outras palavras, o Excel passa direto pelos comentários (').

A primeira declaração VBA (que começa com a palavra Sub) identifica a macro como um procedimento Sub e dá o seu nome — você forneceu esse nome antes de começar a gravar a macro. Se ler cuidadosamente através do código, você será capaz de entender um pouco dele. Você verá o seu nome, a fórmula que forneceu e código adicional que altera a fonte. O procedimento Sub termina com a declaração End Sub.

ExcelVBABokks.indb 29

15/02/2013 13:59:19

30 Parte I: Introdução ao VBA

Ei, eu não gravei isso!

Eu já comentei neste capítulo que gravar que o código gravado exiba todos os tipos uma macro é como gravar som em um de declarações de mudança de fonte gravador de fita. Quando você coloca (Strikethrough, Superscript, Shadow e para tocar e escuta sua própria voz no assim por diante). Não se preocupe, isso gravador, invariavelmente diz: “Minha voz acontece o tempo todo. Frequentemente, não é essa”. E quando você olhara sua o Excel grava muitos códigos que parecem gravação de macro, talvez veja algumas inúteis. Em capítulos posteriores, você ações que não gravou.

verá como remover o código extra de uma

Ao gravar o exemplo NameAndTime, você macro gravada.

mudou apenas o tamanho da fonte, ainda

Modificando a Macro

Como seria esperado, você pode não apenas ver a sua macro em VBE, como também alterá-la. Se olhar para o código, alguma coisa dele fará, de fato, algum sentido. Mesmo que a essa altura você não tenha ideia do que está fazendo, aposto que pode fazer estas mudanças no código: 5 Mudar o nome fornecido na célula ativa. Se você tiver um cachorro, use o nome dele.

5 Mudar o nome da fonte ou do tamanho.

5 Veja se você descobre um lugar adequado para uma nova declaração:

```
Selection.Font.Bold = True
```

Trabalhar com um módulo de código VBA é quase como trabalhar em **DICA**

um documento em um processador de textos (exceto que não há

palavra envolvida e não é possível formatar o texto). Pensando melhor, acho que é mais como trabalhar em Windows Notepad. Você pode

pressionar Enter para iniciar uma nova linha e as teclas de edição conhecidas funcionam conforme esperado.

Depois de ter feito as suas alterações, volte para o Excel e experimente a macro revisada, para ver como ela funciona. Exatamente como você pressiona Alt + F11 em Excel para exibir o VBE, pode pressionar Alt + F11

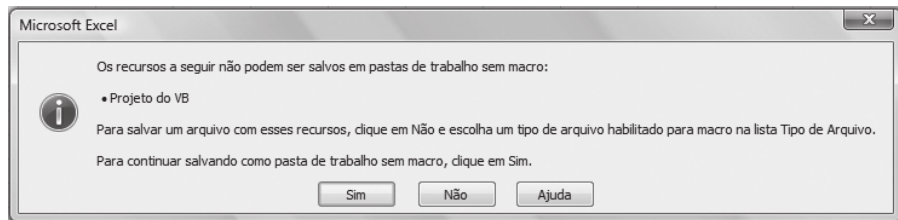
no VBE para voltar ao Excel.

Salvando Planilhas que Contém Macros

Se você armazenar uma ou mais macros na planilha, o arquivo deve ser salvo com macros habilitadas. Em outras palavras, o arquivo deve ser salvo com uma extensão XLSM em lugar da extensão XLSX normal.

ExcelVBABokks.indb 30

15/02/2013 13:59:19



Capítulo 2: Saltando Para o Lugar Certo

31

Por exemplo, quando você salva a planilha que contém a sua macro NameAndTime, o formato de arquivo na caixa de diálogo Salvar Como padroniza para XLSX (um formato que não pode conter macros!). A menos que você mude o formato de arquivo para XLSM, o Excel exibe o aviso mostrado na Figura 2-5. Você precisa clicar em Não e depois, escolher Modelo Habilitado para Macro do Excel (*.xlsm) a partir da lista de seleção do tipo de arquivo do Salvar Como.

Figura 2-5:

Se sua
planilha
contiver
macros e
você tentar
salvá-la em
um formato
de arquivo
sem macro,
o Excel o
avisará.

Entendendo a Segurança de Macro

Em Excel, a segurança de macro é um recurso importante. Pelo fato de VBA ser uma linguagem poderosa – tão poderosa que até mesmo uma simples macro pode causar sérios danos ao seu computador. Uma macro pode apagar arquivos, enviar informações a outros computadores e até destruir o Windows, de modo que você não pode sequer iniciar o seu sistema.

Os recursos de segurança de macro em Excel 2007 e Excel 2010 foram criados para ajudar a evitar tais tipos de problemas

A Figura 2-6 mostra a seção Configurações de Macro da caixa de diálogo Central de Confiabilidade. Para exibir essa caixa de diálogo, escolha Desenvolvedor⇒Código⇒Configurações de Macro.

Por padrão, o Excel usa Desabilitar Todas as Macros com Notificação.

Com essa configuração ativa, se você abrir uma planilha que contenha macros (e o arquivo não esteja digitalmente “assinado” ou armazenado em um lugar confiável), o Excel exibe um aviso como o da Figura 2-7.

Caso você tenha certeza de que a planilha vem de uma fonte confiável, clique Habilitar Macros, e as macros serão habilitadas.

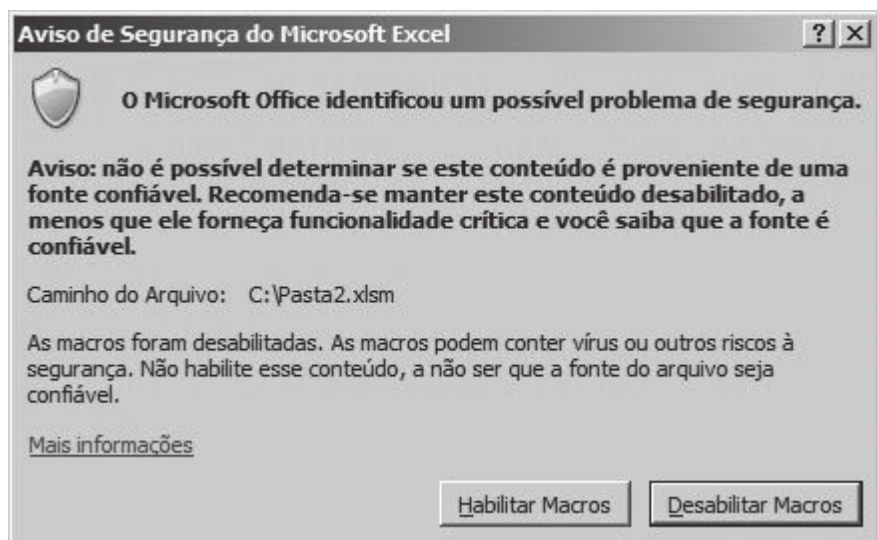
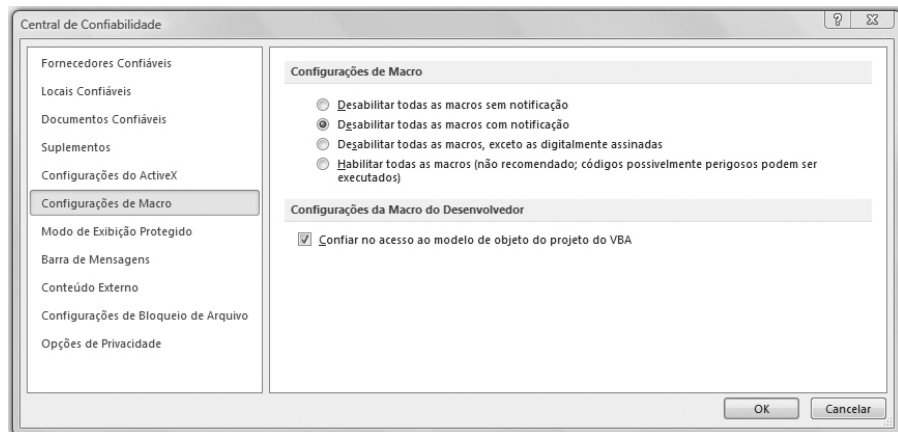
Você só vê a caixa pop-up (que surge) da Figura 2-7 se o VBE estiver **BRE-SE**

aberto. Caso contrário, o Excel exibe uma vista de aviso de segurança **LEM**

(Security Warning) acima da barra Fórmula. É possível clicar o botão para habilitar as macros.

ExcelVBABokks.indb 31

15/02/2013 13:59:19



32 Parte I: Introdução ao VBA

Figura 2-6:

A Seção

Configura-

ções de

Macro da

caixa de

diálogo

Central de

confiabilidade.

Se você usa Excel 2010, ele lembrará se você designou uma planilha para ser protegida. Portanto, na próxima vez que abri-lo, não verá o Aviso de Segurança. Porém, esse não é o caso com Excel 2007. Você receberá um aviso a cada vez — a menos que armazene aquela planilha em um lugar protegido.

Figura 2-7:

O Excel

avisa que

o arquivo

aberto

contém

macros.

Talvez a melhor maneira para lidar com a segurança da macro é designar uma ou mais pastas como locais confiáveis. Todas as planilhas em um local confiável são abertas sem um aviso da macro. As pastas confiáveis são designadas na seção Locais Confiáveis, na caixa de diálogo Central de Confiabilidade.

Querendo descobrir o que significam as outras configurações de segurança, pressione F1 enquanto a seção Configurações de Macro da caixa de diálogo Central de Confiabilidade estiver à vista. A tela Ajuda do Excel aparece e o tópico Habilitar ou Desabilitar Macros em Arquivos do Office é exibido na janela Ajuda do Excel.

ExcelVBABokks.indb 32

15/02/2013 13:59:19

Capítulo 2: Saltando Para o Lugar Certo

33

Mais sobre a Macro NameAndTime

Quando você terminar este livro, entenderá completamente como a macro NameAndTime funciona e estará preparado para desenvolver macros. Por ora, mostro o exemplo com alguns pontos adicionais sobre a macro: 5 Para esta macro funcionar, a planilha deve estar aberta. Se você fechar a planilha, a macro não funciona (e o atalho Ctrl + Shift + N

não terá efeito).

5 Desde que a planilha contendo a macro esteja aberta, você pode rodar a macro enquanto qualquer planilha estiver ativa. Em outras palavras, a planilha da própria macro não precisa estar ativa.

5 A macro não é código ‘pró qualidade’. Ela sobrescreverá texto existente sem aviso – e os seus efeitos não podem ser desfeitos.

5 Antes de começar a gravar a macro, você designou a ela uma nova tecla de atalho. Essa é apenas uma de várias maneiras de executar a macro.

5 Você pode entrar manualmente com essa macro, ao invés de gravá-la. Para fazê-lo, é necessário um bom entendimento de VBA (tenha paciência, você chegará lá).

5 É possível armazenar essa macro em sua Pasta de trabalho de macros pessoais. Se o fizer, a macro estará automaticamente

disponível sempre que você iniciar o Excel. Para detalhes sobre isso, veja o Capítulo 6.

5 Também é possível converter a planilha a um arquivo add-in

(mais sobre isso no Capítulo 21).

Você foi iniciado no mundo de programação do Excel. Lamento, não há um aperto de mão ou anel decodificador. Espero que este capítulo o ajude a perceber que programação em Excel é algo que você realmente pode fazer — e até viver para contar. Quase que com certeza, os próximos capítulos respondem às suas perguntas, e logo você entenderá exatamente o que fez nesta seção prática.

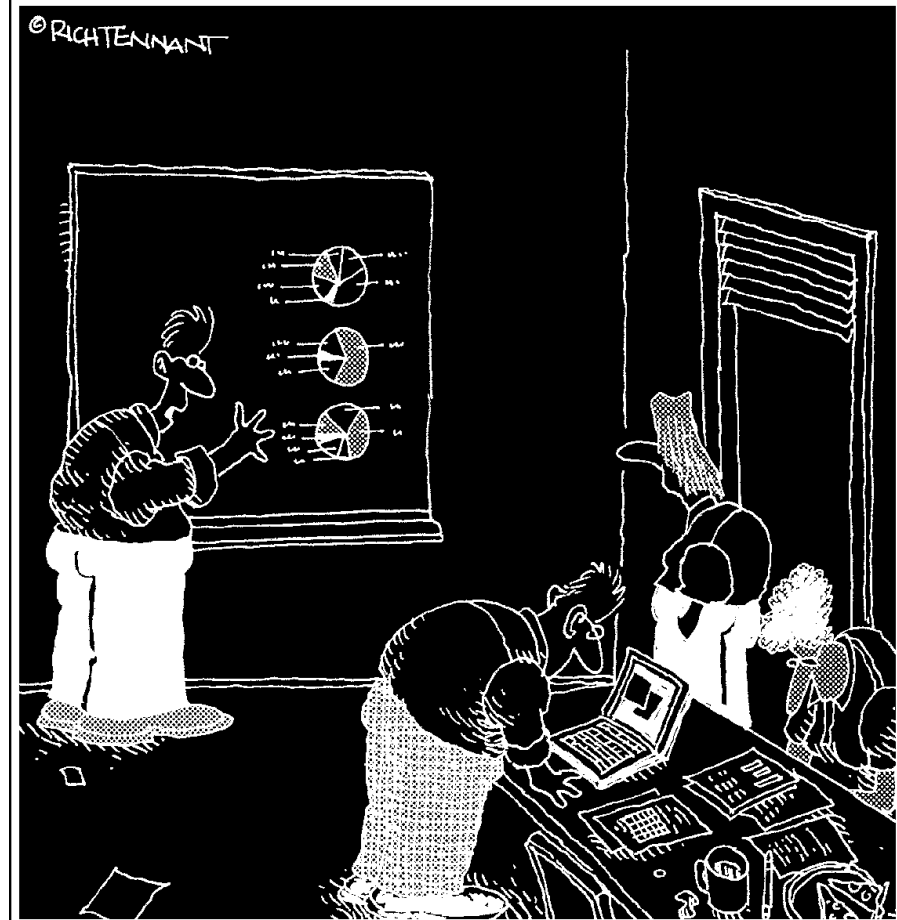
ExcelVBABokks.indb 33

15/02/2013 13:59:19

34 Parte I: Introdução ao VBA

ExcelVBABokks.indb 34

15/02/2013 13:59:19



Parte II

Como o VBA Trabalha

com o Excel

A 5ª Onda

Por Rich Tennant

ESTRATÉGIA PARA

AUMENTO DE

LUCRO

“Veja – e se apenas aumentássemos o

tamanho dos gráficos?”

ExcelVBABokks.indb 35

15/02/2013 13:59:19

Nesta parte...

O s próximos quatro capítulos oferecem uma base importante para descobrir os prós e os contras do VBA. Você aprenderá sobre módulos (as planilhas que armazenam o seu código VBA) e será apresentado ao modelo objeto Excel (algo que você não quer perder). Você também descobrirá a diferença entre sub-rotinas e funções, e terá um curso relâmpago sobre o gravador de macro do Excel.

ExcelVBABokks.indb 36

15/02/2013 13:59:19

Capítulo 3

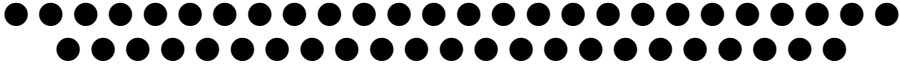
Trabalhando no Visual

Basic Editor



Neste Capítulo

- ☐ Entenda o Visual Basic Editor
- ☐ Descubra as partes do Visual Basic Editor
- ☐ Saiba o que acontece em um módulo VBA
- ☐ Entenda as três maneiras de ter código VBA em um módulo
- ☐ Personalize o ambiente VBA



Como um usuário mais experiente do que a média em Excel, provavelmente você tem uma boa ideia sobre pastas de trabalhos, fórmulas, tabelas e outras benesses do Excel. Agora, é hora de expandir seus horizontes e explorar um aspecto totalmente novo de Excel: o Visual Basic Editor. Neste capítulo, você descobrirá como trabalhar com ele e vai direto ao que interessa quanto a escrever algum código VBA.

O Que É o Visual Basic Editor?

Eu vou poupar meus dedos do preâmbulo e me referir ao Visual Basic Editor como o VBE. O VBE é um aplicativo separado, onde você escreve e edita suas macros VBA. Ele funciona sem emendas com o Excel. Por *sem emendas* quero dizer que o Excel cuida de abrir o VBE quando você precisa dele.

BRE-SE

Não é possível rodar VBE separadamente; o Excel precisa estar rodando **LEM**

para que o VBE seja executado.

Ativando o VBE

O modo mais rápido de ativar o VBE é pressionando Alt + F11 quando o Excel está ativo. Para retornar ao Excel, pressione Alt + F11 de novo.

Você também pode ativar o VBE usando o comando

Desenvolvedor ⇨ Código ⇨ Visual Basic. Se você não tem uma faixa de opções Desenvolvedor na parte superior da sua janela Excel, volte ao Capítulo 2, onde explico como obtê-la .

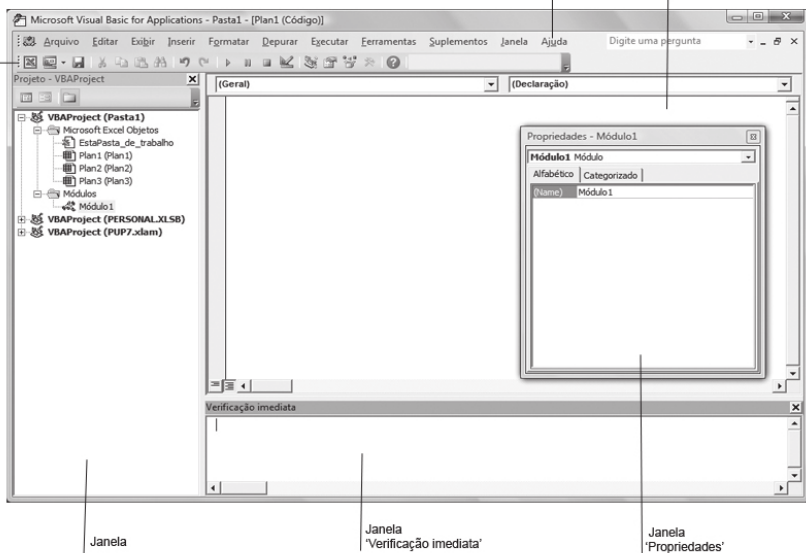
ExcelVBABokks.indb 37

15/02/2013 13:59:19

Barra
de Ferramentas

Barra
de Menu

Janela
de Código



38 Parte II: Como o VBA Trabalha com o Excel

Entendendo os componentes do VBE

A Figura 3-1 mostra o programa, com algumas das partes chave identifi-DICA

cadas. Pelo fato de haver tantas coisas em andamento no VBE, eu gosto de maximizar a janela para ver tanto quanto possível.

É possível que a sua janela de programa VBE não se pareça exatamente com o que é visto na Figura 3-1. O VBE contém várias janelas e é altamente personalizável. Você pode ocultar, reorganizar, acoplar janelas e assim por diante.

Figura 3-1:

O VBE é

seu amigo

personalizável.

Na verdade, o VBE possui ainda mais partes do que as mostradas na Figura 3-1. Eu discutirei esses componentes adicionais no decorrer do livro, quando eles se tornarem relevantes.

A barra de menu do VBE funciona como qualquer outra barra de menu que você já tenha encontrado. Ela contém comandos que são usados para realizar tarefas com os diversos componentes no VBE. Você descobre ainda que muitos dos comandos de menu têm teclas de atalho associadas.

ExcelVBABokks.indb 38

15/02/2013 13:59:20

Capítulo 3: Trabalhando no Visual Basic Editor

39

O VBE também tem menus de atalho. Você pode clicar com o botão **DICA**

direito em praticamente qualquer coisa no VBE e obter um menu de atalho de comandos comuns.

Barra de Ferramentas

A barra de ferramentas Padrão, que está abaixo da barra de menu (veja a Figura 3-1), é uma das quatro barras de ferramentas de VBE disponíveis.

vel. Você pode personalizar as barras de ferramentas, movê-las, mostrar outras barras de ferramentas e assim por diante. Se você quiser, use o comando Exibir ⇨ Barras de Ferramentas para trabalhar com as barras de ferramentas de VBE. A maioria das pessoas (inclusive eu) apenas as deixa como são.

Janela Projeto

A janela Projeto exibe um diagrama em árvore que apresenta toda a planilha aberta em Excel no momento (inclusive add-ins e planilhas ocultas). Clique duas vezes nos itens para expandi-los ou recolhê-los.

Eu forneço mais detalhes sobre essa janela na próxima seção “Como trabalhar com a Janela Projeto”.

Se a janela Projeto não estiver visível, pressione Ctrl + R ou use o comando Exibir⇨Project Explorer. Para ocultar a janela Projeto,

clique com botão Fechar em sua barra de título. Ou clique com o botão direito em qualquer lugar na janela Projeto e selecione Ocultar a partir do menu de atalho.

Janela de Código

Uma janela de Código contém o código VBA. Cada objeto em um projeto tem uma janela de Código associada. Para ver a janela de Código de um objeto, clique duas vezes no objeto na janela Projeto. Por exemplo, para ver a janela de Código do objeto Plan1, clique duas vezes Plan1 na janela Projeto. A menos que você tenha acrescentado algum código VBA, a janela de Código estará vazia.

Você encontrará mais sobre janelas de Código mais adiante neste capítulo, na seção Trabalhando com a janela de Código.

Janela Verificação Imediata

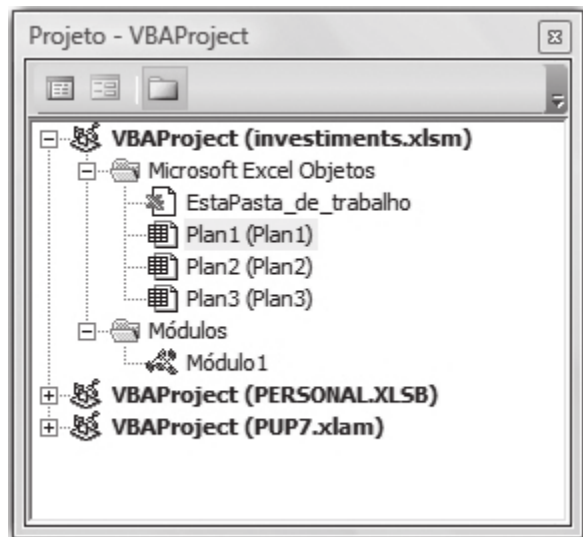
A janela Verificação Imediata pode ou não estar visível. Se não estiver visível, pressione Ctrl + G ou use o comando Exibir⇨Janela Verificação Imediata. Para fechar a janela Verificação Imediata, clique o botão Fechar na barra de título (ou clique com o botão direito em qualquer lugar na janela Verificação Imediata e selecione Ocultar do menu de atalho).

A janela Verificação Imediata é mais útil para executar diretamente declarações VBA e para depurar o seu código. Se você estiver apenas começando em VBA, essa janela não será tão útil, portanto, fique à vontade para ocultá-la e liberar algum espaço na tela para outras coisas.

No Capítulo 13, discuto em detalhes a janela Verificação Imediata. Ela pode se tornar sua boa amiga!

ExcelVBABokks.indb 39

15/02/2013 13:59:20



40 Parte II: Como o VBA Trabalha com o Excel

O que há de novo no Visual Basic Editor?

O Excel 2007 apresentou uma nova interface. A linguagem de programação VBA foi atualizada de usuário. Menus e barras de ferramentas foram substituídas pelas novas do Excel, mas o VBE não tem novas funcionalidades. Se você já usou o Visual Basic Editor e as barras de ferramentas e menus de uma versão anterior do Excel, estará em estilo antigo. Funcionam exatamente como território familiar. No Office 2007, a Microsoft sempre funcionaram. Talvez eles tenham deixado o VBE essencialmente intacto. E atualizar o VBE, mas não estou entendendo manteve a tradição no Excel 2010.

a respiração.

Como Trabalhar com a Janela de Projeto

Quando você trabalha no VBE, cada pasta de trabalho do Excel e add-in que está aberta é um projeto. É possível pensar em projeto como uma coleção de objetos dispostos como um esboço. Você pode expandir um projeto clicando no sinal de adição (+) à esquerda do nome do projeto na janela Projeto. Contraia um projeto clicando no sinal de subtração (-) à esquerda do nome do projeto. Ou você pode dar dois cliques nos itens para expandir e contraí-los.

A Figura 3-2 mostra uma janela Project com três projetos relacionados: um add-in chamado pup7.xlam, uma pasta de trabalho

chamada

investments.xlsm e a Pasta de Trabalho Pessoal de Macros (que é sempre chamada de PERSONAL.XLSB).

Figura 3-2:

Esta janela
do Projeto
relaciona
três projetos.
Um deles é
expandido
para exibir
seus objetos.

Cada projeto expande para mostrar no mínimo um nó chamado

Microsoft Excel Objects. Este nó expande para mostrar um item para cada planilha na pasta de trabalho (cada planilha é considerada um objeto), e outro objeto chamado ThisWorkbook (que representa o objeto Workbook). Se o projeto tiver quaisquer módulos VBA, a listagem de projeto também exibe os nós referentes a esses módulos. E, ExcelVBABokks.indb 40

15/02/2013 13:59:20

Capítulo 3: Trabalhando no Visual Basic Editor

41

como pode ser visto na Parte IV, um projeto também pode conter um nó chamado Forms (formulários), o qual contém objetos UserForm (que mantém caixas de diálogo personalizadas).

O conceito de objetos pode estar um pouco vago para você. Entretanto, garanto que as coisas ficarão mais claras nos capítulos seguintes. Não se preocupe muito se não entender o que está

acontecendo neste ponto.

Adicionando um novo módulo VBA

Siga esses passos para adicionar um novo módulo VBA ao projeto: **1. Selecione o nome do projeto na janela de Projeto**

2. Escolha Inserir⇒Módulo

Ou

1. Clique com o botão direito no nome do projeto

2. Escolha Inserir⇒Módulo do menu de atalho

Quando você grava uma macro, automaticamente o Excel insere um mó-

DICA

dulo VBA para conter o código gravado. Qual pasta de trabalho contém o módulo da macro gravada depende de onde você decide armazenar a macro gravada, pouco antes de começar a gravar.

Removendo um módulo VBA

Precisa remover um módulo VBA de um projeto?

1. Selecione o nome do módulo na janela Projeto

2. Escolha Arquivo⇒Remover xxx, onde xxx é o nome do módulo

Ou

1. Clique com o botão direito no nome do módulo

2. Escolha Remover xxx do menu de atalho

O Excel, sempre tentando evitar que você faça alguma coisa da qual se arrependerá, perguntará se você quer exportar o código do módulo antes de apagá-lo. Quase sempre, não (se quer exportar o módulo, veja a próxima seção).

ExcelVBABokks.indb 41

15/02/2013 13:59:20

42 Parte II: Como o VBA Trabalha com o Excel Você pode remover

módulos VBA, mas não tem jeito de remover os outros módulos de código — aqueles para os objetos Sheet ou ThisWorkbook.

Exportando e importando objetos

Todo objeto em um projeto VBA pode ser salvo em um arquivo separado. Salvar um objeto individual de um projeto é conhecido como exportação. É lógico que você também pode importar objetos para um projeto. Exportar e importar objetos pode ser útil se quiser usar um objeto em especial (tal como um módulo VBA ou um UserForm) em um projeto diferente.

Siga os seguintes passos para exportar um objeto:

- 1. Selecione um objeto na janela Projeto**
- 2. Escolha Arquivo⇒Exportar Arquivo ou pressione Ctrl + E**

Você obtém uma caixa de diálogo que pede um nome de arquivo.

Observe que o objeto permanece no projeto; apenas uma cópia dele é exportada.

Importar um arquivo para um projeto é assim:

- 1. Selecione o nome do projeto na janela Projeto.**
- 2. Escolha Arquivo⇒Importar Arquivo ou pressione Ctrl + M.**

BRE-SE

Você obtém uma caixa de diálogo que pede um arquivo. Locali-

LEM

ze o arquivo e clique em Abrir. Você só deve importar um arquivo se o arquivo foi exportado usando o comando Arquivo ⇒ Exportar de Código.

Trabalhando com a Janela de Código

Enquanto você se transforma num especialista em VBA, passa muito **BRE-SE**

tempo trabalhando nas janelas Código. As macros que você grava

estão **LEM**

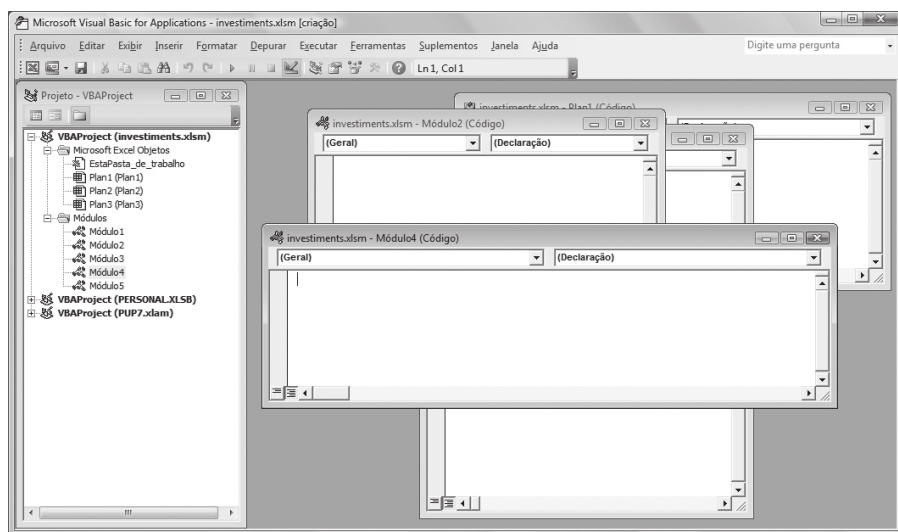
armazenados em um módulo e você pode digitar o código VBA diretamente em um módulo VBA.

Minimizando e maximizando janelas

Se você tem vários projetos abertos, o VBE deve ter muitas janelas de código abertas em determinada ocasião. A Figura 3-3 mostra um exemplo do que eu quis dizer.

ExcelVBABokks.indb 42

15/02/2013 13:59:20



Capítulo 3: Trabalhando no Visual Basic Editor

43

Figura 3-3:

A janela
de Código
sobrecar—
regada

não é

bonita.

As janelas de código são muito parecidas com as janelas da pasta de trabalho no Excel. Você pode minimizá-las, maximizá-las, ocultá-las, reajustá-las e assim por diante. A maioria das pessoas acha muito mais fácil maximizar a janela de código na qual estão trabalhando. Fazer isto permite ver mais código e impede que você se distraia.

Para maximizar uma janela de código, clique no botão Maximizar na sua barra de título (perto do X). Ou apenas clique duas vezes a sua barra de título para maximizá-la. Para recuperar uma janela de código ao seu tamanho original, clique no botão Restaurar. Quando uma janela é maximizada, a sua barra de título não é visível, portanto, você encontrará o botão Restaurar abaixo da barra de título de VBE.

Às vezes, você pode querer ter duas ou mais janelas de código visíveis.

Por exemplo, você quer comparar o código de dois módulos ou copiar o código de um módulo para outro. É possível organizar as janelas manualmente, usar os comandos Janela⇒Lado a lado Horizontal ou Janela⇒Lado a lado Vertical para organizá-las automaticamente.

Você pode alternar rapidamente para trás e para frente entre as janelas de código, pressionando Ctrl + Tab. Se repetir essa combinação de teclas, manterá um ciclo por todas as janelas de código abertas.

Pressionar Ctrl + Shift + Tab permite voltar na ordem inversa.

ExcelVBABokks.indb 43

15/02/2013 13:59:20

44 Parte II: Como o VBA Trabalha com o Excel Minimizar uma janela de Código fica fora de mão. Você também pode **DICA**

clique o botão Fechar da janela (que exibe “X”) em uma barra de título da janela de Código, para fechá-la completamente (fechar a janela apenas a oculta; você não perde nada). Para abri-la de novo, apenas dê dois cliques no objeto apropriado na janela Projeto. Trabalhar com essas janelas de código parece mais difícil do que é realmente.

Criando um módulo

Em geral, um módulo pode manter três tipos de código:

5 Declarações: Uma ou mais declarações de informação que você fornece para o VBA. Por exemplo, é possível declarar o tipo de dados para variáveis que você pretende usar ou configurar outras opções relacionadas ao módulo.

5 Procedimentos Sub: Um conjunto de instruções de programação que executa alguma ação.

5 Procedimento Function: Um conjunto de instruções de programação que retorna um valor único (similar ao conceito de uma função de planilha, como SOMA).

Um único módulo VBA pode armazenar qualquer quantidade de procedimentos Sub, procedimentos Function e declarações. Bem, há um limite — cerca de 64.000 caracteres por módulo. Como forma de comparação, este capítulo em especial tem aproximadamente a metade de tal quantidade de caracteres. Depois de mais de 15 anos de programação VBA, eu nem ao menos cheguei perto de atingir aquele limite. E

se eu o fiz, a solução é simples: apenas insira um novo módulo.

A organização de um módulo VBA só depende de você. Algumas pessoas preferem manter todo o código VBA para um aplicativo em um único módulo VBA, outras gostam de separar o código em vários módulos diferentes. É uma escolha pessoal, como arrumar os móveis.

Como inserir código VBA em um módulo

Um módulo VBA vazio é como a comida falsa que você vê nas janelas de alguns restaurantes chineses; parece boa, mas na verdade não faz muito por você. Antes de poder fazer algo significativo, você deve ter algum código VBA no módulo. É possível inserir o código VBA em um módulo de três formas:

5 Inserir o código diretamente.

5 Usar o gravador de macro do Excel para gravar suas ações e convertê-las em código VBA (veja o Capítulo 6).

5 Copiar o código de um módulo e colar em outro.

ExcelVBABokks.indb 44

15/02/2013 13:59:20

Capítulo 3: Trabalhando no Visual Basic Editor

45

DE ESPECIALISTA

O P A P

Pausa para um intervalo de terminologia

Preciso desviar o assunto por um momento Function — sendo que, às vezes, ambos são para discutir terminologia. Neste livro, uso chamados de rotinas. Uso todos esses termos procedimentos Sub, rotina, promos de modo permutável. No entanto, como cedimento e macro. Esses termos são um será detalhado nos capítulos seguintes, há pouco confusos. Colegas de programação uma diferença importante entre os procedi-geralmente usam a palavra procedimento mentos Sub e Function. Por ora, não se preo-para descrever uma tarefa automatizada. cupe com a terminologia. Apenas tente en-Tecnicamente, um procedimento pode se tender os conceitos.

referir a um Sub ou a um procedimento

Inserindo o código diretamente

Às vezes, o melhor caminho é o mais direto. Inserir o código diretamente implica em bem, inserir o código diretamente. Em outras palavras, você digita o código pelo seu teclado. Insere e edita o texto em um módulo VBA que funciona como deveria. Você pode selecionar, copiar, cortar, colar e fazer outras coisas com o texto.

Use a tecla Tab para alinhar algumas das linhas para tornar seu código mais fácil de ler. Isto não é necessário, mas é um bom hábito. Conforme estuda o código que apresento neste livro, você entenderá porque é útil alinhar linhas de código do parágrafo.

Uma única linha do código de VBA pode ser do tamanho que você
BRE-SE

quiser. Entretanto, você pode querer usar o caractere de continuação
LEM

de linha para fragmentar linhas longas de código. Para continuar uma única linha de código (também conhecida como uma declaração) de uma linha para a próxima, termine a primeira linha com um espaço seguido de um underline (_). Então, continue a declaração na próxima linha. Eis um exemplo de sentença única, separada em três linhas: Selection.Sort Key1: = Range("A1"), _

Order1: = xlAscending, Header: = xlGuess, _

Orientation: = xlTopToBottom

Esta declaração executaria da mesma maneira, como se você tivesse inserido em uma única linha (sem caracteres de continuação da linha).

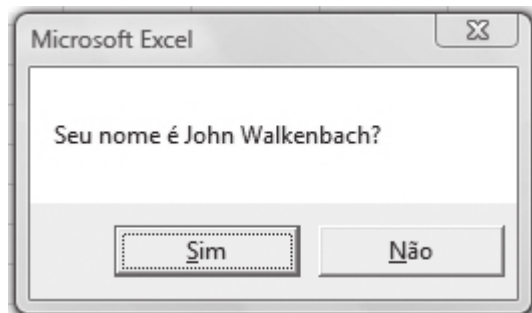
Veja que eu alinhei a segunda e a terceira linhas dessa declaração. O

alinhamento é opcional, mas deixa claro que essas linhas não são declarações separadas.

Os engenheiros de colarinho branco que projetaram o VBE perceberam **DICA**

que pessoas como nós cometeriam erros. Portanto, o VBE tem múltiplos níveis de desfazer e refazer. Se você apagou uma declaração que não deveria, use o botão Desfazer da barra de ferramentas (ou pressione Ctrl + Z) até que a declaração reapareça. Depois de desfazer, você ExcelVBABokks.indb 45

15/02/2013 13:59:20



46 Parte II: Como o VBA Trabalha com o Excel

pode usar o botão Refazer para retomar as mudanças que tinha desfeito. Esse negócio de desfazer/refazer é mais complicado para descrever do que para usar. Recomendo brincar um pouco com esse recurso até entender como ele funciona.

Pronto para inserir algum código de verdade? Tente os seguintes passos: **1. Crie uma nova pasta de trabalho no Excel**

2. Pressione Alt + F11 para ativar o VBE

3. Clique o nome da pasta de trabalho na janela Projeto

4. Escolha Inserir⇒Módulo para inserir um módulo VBA no projeto **5. Digite o seguinte código no módulo:**

```
Sub GuessName()
```

```
Msg = "Seu nome John Walkenbach? " & Application.UserName &  
"?"
```

```
Ans = MsgBox(Msg, vbYesNo)
```

```
If Ans = vbNo Then MsgBox "Nao se preocupe."
```

```
If Ans = vbYes Then MsgBox "Eu devo ser  
adivinho!"
```

```
End Sub
```

6. Verifique se o cursor está em qualquer lugar no texto que você digitou e pressione F5 para executar o procedimento.

F5 é um atalho para o comando Executar⇒Executar Sub/

UserForm. Se você inseriu o código corretamente, o Excel executa o procedimento e você pode responder a caixa de diálogo

mostrada na Figura 3-4. A menos que o seu nome seja igual ao meu, a caixa de diálogo será diferente daquela que aparece na figura.

Figura 3-4: O

procedimento

GuessName

aparece

nessa caixa

de diálogo.

Quando você entra com o código listado no Passo 5, deve perceber que o VBE faz alguns ajustes no texto que foi inserido. Por exemplo, depois de você digitar a declaração Sub, automaticamente o VBE insere a declaração End Sub. E se você omitir o espaço antes ou depois um sinal de igual, o VBE insere o espaço para você. Além disso, ele altera a cor e as letras maiúsculas de algum texto. Isto tudo é perfeitamente normal. É

apenas a maneira pela qual o VBE mantém as coisas claras e legíveis.

Se você acompanhou os passos anteriores, já escreveu um procedimento Sub do VBA, também conhecido como macro. Quando você pressiona F5, o Excel executa o código e segue as instruções. Em outras palavras, o Excel avalia cada sentença e faz o que foi pedido para fazer (não deixe que este poder recém descoberto suba à sua cabeça).Você ExcelVBABokks.indb 46

15/02/2013 13:59:20

Capítulo 3: Trabalhando no Visual Basic Editor

47

pode executar essa macro quantas vezes quiser, embora ela tenda a perder o seu encanto depois de usada algumas dezenas de vezes.

Só para ficar registrado, essa simples macro usa os seguintes conceitos, sendo todos eles abordados neste livro:

5 Definir um procedimento Sub (a primeira linha)

5 Designar valores para variáveis (Msg e Ans)

5 Concatenar (juntar) uma string (usando o operador &)

5 Usar uma função VBA integrada (MsgBox)

5 Usar constantes VBA integradas (vbYesNo, vbNo e vbYes)

5 Usar uma construção If-Then (duas vezes)

5 Finalizar um procedimento Sub (a última linha)

Nada mal para um iniciante, não é?

Usando o gravador de macro

Outra maneira de você colocar o código em um módulo VBA é gravar suas ações, usando o gravador de macro do Excel. Se você trabalhou com todo o exercício prático do Capítulo 2, já deve ter alguma experiência com essa técnica.

A propósito, não há como poder gravar o procedimento `GuessName` **BRE-SE**

mostrado na seção anterior. Só é possível gravar o que pode ser feito **LEM**

diretamente no Excel. Exibir a caixa de mensagem não é um repertório normal do Excel (isso é uma coisa VBA). O gravador de macro é útil,

mas, em muitos casos, provavelmente você precisará inserir manualmente no mínimo algum código.

Eis um exemplo, passo a passo, que mostra como gravar uma macro que desativa as grades de linha da célula em uma planilha. Se quiser tentar este exemplo, abra uma nova pasta de trabalho, e siga estes passos: **1. Ative uma planilha na pasta de trabalho** Qualquer planilha servirá. Se a planilha não está mostrando as linhas de grades, adicione uma nova planilha que o faça. Você precisa começar com uma planilha que tenha linhas de grades.

2. Selecione Desenvolvedor⇒Código⇒Gravar Macro. Ou você pode clicar no ícone com um pequeno ponto vermelho no lado esquerdo da barra de status.

O Excel exibe sua caixa de diálogo Gravar Macro.

3. Na caixa de diálogo Gravar Macro, nomeie a macro como Gridlines e use Ctrl + Shift + F como a tecla de atalho.

ExcelVBABokks.indb 47

15/02/2013 13:59:20

48 Parte II: Como o VBA Trabalha com o Excel 4. Clique em OK para começar a gravar.

O Excel insere automaticamente um novo módulo VBA no projeto que corresponde à pasta de trabalho ativa. A partir deste ponto, o Excel converte suas ações em código VBA. Enquanto grava, o

ícone na barra de status se transforma em um pequeno quadrado azul. Isto é um lembrete de que o gravador de macro está rodando. Você também pode clicar naquele quadrado azul para parar a gravação da macro.

5. Selecione Exibição⇒Mostrar⇒Linhas de Grade.

As linhas de grades na planilha desaparecem.

6. Selecione Desenvolvedor⇒Código⇒Parar Gravação. Ou clique no botão Parar Gravação na barra de status (o quadrado azul).

O Excel interrompe a gravação de suas ações.

Para visualizar esta macro, gravada recentemente, pressione Alt + F11

para ativar o editor VB. Localize o nome da pasta de trabalho na janela do Project Explorer. Veja que o projeto tem um novo módulo listado. O

nome do módulo depende se você tinha outros módulos na pasta de trabalho quando iniciou a gravação do macro. Se não, o módulo será nomeado como Módulo1. Você pode dar dois cliques no módulo para visualizar a sua janela de código.

Aqui está o código gerado pelas suas ações:

```
Sub Gridlines()
```

```
“ Gridlines Macro
```

```
‘
```

```
‘ Keyboard Shortcut: Ctrl + Shift + G
```

```
‘ ActiveWindow.DisplayGridlines = False
```

```
End Sub
```

Para testar essa macro, ative uma planilha que exiba linhas de grades e depois pressione a tecla de atalho que você designou no Passo 3: Ctrl + Shift + F.

Se não designou uma tecla de atalho para a macro, não se preocupe. Aqui está como mostrar uma lista de todas as macros e rodar aquela que quiser.

1. Selecione Desenvolvedor⇒Código⇒Macros.

Os fãs do teclado podem pressionar Alt + F8. Qualquer desses métodos exibe uma caixa de diálogo que relaciona todas as macros disponíveis.

2. Selecione a macro na lista (neste caso, Gridlines).

3. Clique no botão Executar.

O Excel executa a macro e, magicamente, as linhas de grade

desaparecem.

ExcelVBABokks.indb 48

15/02/2013 13:59:20

Capítulo 3: Trabalhando no Visual Basic Editor

49

Lógico que você pode executar qualquer quantidade de comandos e ações enquanto o gravador de macro estiver rodando. O Excel traduz fielmente as suas ações do mouse e toques de teclado para o código VBA. Isso funciona como um gravador de fita, exceto pelo Excel nunca ficar sem fita.

Na verdade, essa macro não é tão útil. Afinal, é bem fácil desabilitar as linhas de grade sem uma macro. Seria mais útil se ela pudesse alternar, ativando e desativando, as linhas de grades. Para mudar isso, ative o módulo e troque a declaração para:

```
ActiveWindow.DisplayGridlines = _
```

```
Not ActiveWindow.DisplayGridlines
```

Essa modificação faz com que a macro funcione como um alternador. Se as linhas de grades são exibidas, a macro as desativa. Ops, estou exagerando — desculpe, mas não pude impedir esse exagero. A propó-

sito, esse é outro exemplo de macro que não pode ser gravado. Você pode gravar uma que ative ou desative as linhas de grade, mas não pode gravar uma que irá alterá-las.

Copiando o código VBA

O método final para obter o código no módulo VBA é copiá-lo de outro módulo. Ou de algum outro lugar (tal como um site). Por exemplo, um procedimento Sub ou função que você escreve para um projeto também pode ser útil em outro projeto. Ao invés de perder tempo inserindo de novo o código, pode-se ativar o módulo e usar os procedimentos de copiar e colar da Área de Transferência (eu prefiro mais os atalhos de teclado, Ctrl + C para copiar e Ctrl + V para colar). Depois de colar no módulo VBA, você pode modificar o código se necessário.

Também é possível encontrar muitos exemplos de código VBA na Web.

Se você quiser experimentá-los, selecione o código em seu browser e pressione Ctrl + C para copiá-lo. Depois, ative um módulo e pressione Ctrl + V para colá-lo.

Personalizando o Ambiente VBA

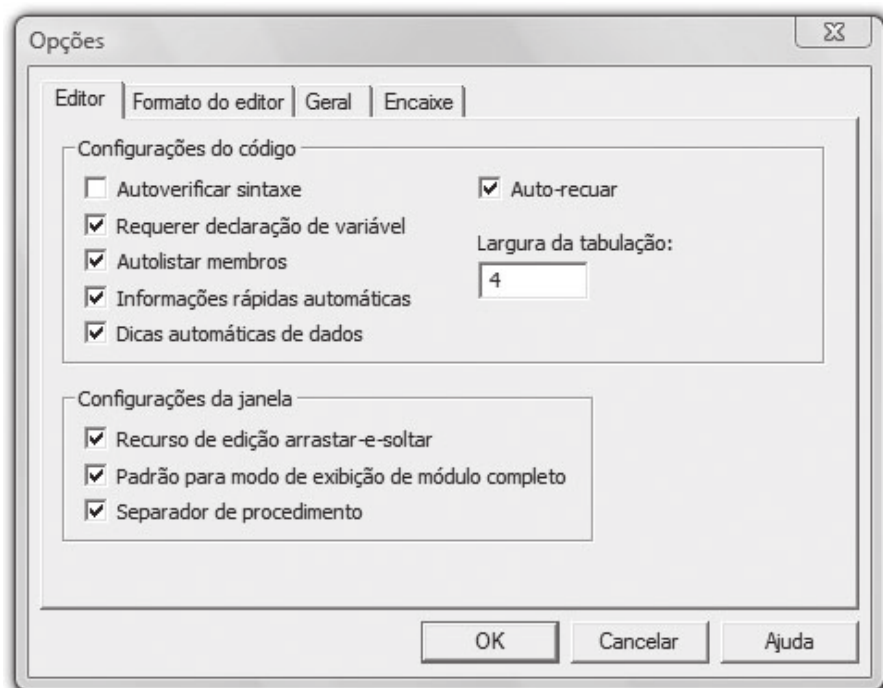
Se está pensando em ser um programador de Excel, gastará muito tempo com os módulos VBA na sua tela. Para ajudar a tornar as coisas tão confortáveis quanto possível (não, por favor, coloque seus pés no chão), o editor VB fornece algumas poucas opções de personalização.

Quando o editor VBE estiver ativado, selecione Ferramentas => Opções.

Você verá uma caixa de diálogo com quatro tabulações: Editor, Formato do Editor, Geral e Encaixe. Discuto algumas das opções mais úteis, nas seções que seguem.

ExcelVBABokks.indb 49

15/02/2013 13:59:21



Usando a guia Editor

A Figura 3-5 mostra as opções disponíveis na guia Editor da caixa de diálogo Opções. Use as opções dessa guia para controlar como certas coisas funcionam no VBE.

Figura 3-5:

Esta é a

guia Editor

na caixa

de diálogo

Opções.

Opção de Autoverificar sintaxe

A opção Autoverificar Sintaxe determina se o VBE exibe uma caixa de diálogo ou se ele encontra um erro de sintaxe enquanto você está inserindo o seu código VBA. A caixa de diálogo informa mais ou menos qual é o problema. Se você não escolher essa opção, o VBE assinala os erros de sintaxe exibindo-os com uma cor diferente em relação ao resto do código, e você não precisa lidar com quaisquer caixas de diálogo aparecendo em sua tela.

Geralmente, mantenho essa configuração desativada porque acho as caixas de diálogo irritantes e geralmente posso descobrir o que está errado com a declaração. Antes de ser um veterano de VBA, achei esse auxílio muito útil.

Opção Requerer declaração de variável

Se a opção Requerer declaração de variável estiver configurada, o VBE insere a seguinte sentença no início de cada novo módulo VBA que você insere:

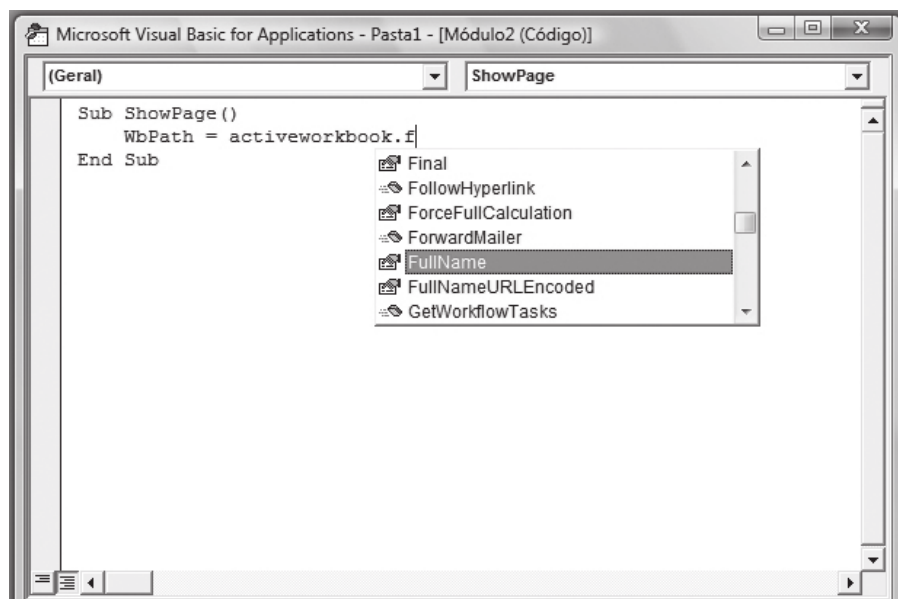
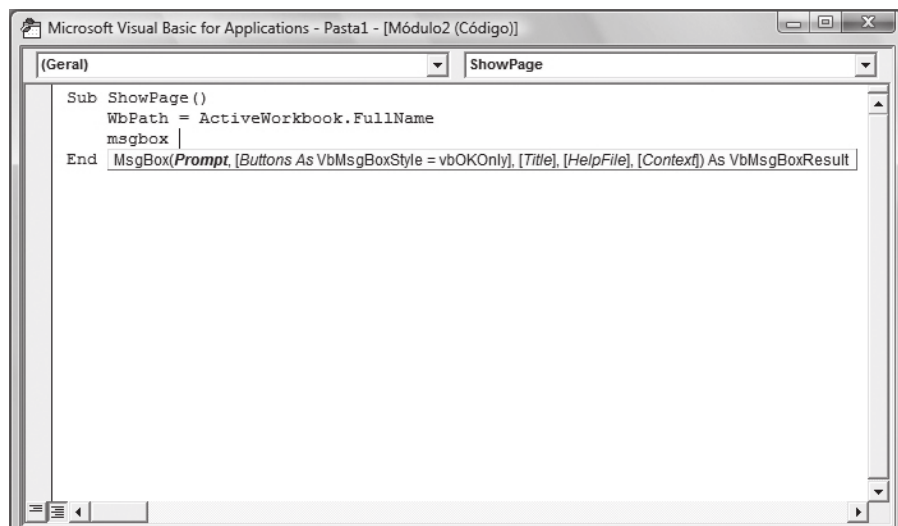
Option Explicit

Mudar essa configuração afeta apenas novos módulos, módulos não existentes. Se essa sentença aparece no seu módulo, você deve definir explicitamente cada variável que usa. No Capítulo 7, explico porque

você deve desenvolver esse hábito.

ExcelVBABokks.indb 50

15/02/2013 13:59:21



Capítulo 3: Trabalhando no Visual Basic Editor

Opção Autolistar membros

Se a opção Autolistar membros estiver configurada, o VBE fornece algum auxílio quando estiver inserindo o seu código VBA. Ela exibe uma lista que completaria logicamente a sentença que você está digitando.

Esse pedaço de mágica é algo chamado de “IntelliSense” (sentido de inteligência).

Esse é o melhor dos recursos do VBE e eu o deixo sempre ativado. A Figura 3-6 mostra um exemplo (que fará muito mais sentido quando você começar a escrever o código VBA).

Figura 3-6:

Um exemplo
de listagem
automática
de membros

Opção Informações rápidas automáticas

Se a opção Informações Rápidas Automáticas estiver configurada, o VBE

exibe informações sobre funções e seus argumentos quando você digitar. Isso pode ser muito útil. A Figura 3-7 mostra esse recurso em ação, informando sobre os argumentos para a função MsgBox.

Opção Dicas automáticas de dados

Se a opção Dicas Automáticas de Dados estiver configurada, o VBE exibe o valor da variável sobre a qual o seu cursor está colocado quando você estiver depurando o código. Ao entrar no maravilhoso mundo da depura-

ção, conforme descrevo no Capítulo 13, você gostará dessa opção.

Figura 3-7:

A opção
Informa-

ções

rápidas

automáticas

oferece

ajuda sobre

a função

MsgBox.

ExcelVBABokks.indb 51

15/02/2013 13:59:21

52 Parte II: Como o VBA Trabalha com o Excel *Configuração Auto-recuar*

A configuração de Auto-recuar determina se o VBE faz automaticamente um recuo em cada linha nova do código como na linha anterior. Eu sou ótimo usando recuos em meu código, então mantenho essa opção ativada.

Use a tecla Tab para recuar seu código, não a barra de espaço. Além **DICA**

disso, você pode usar Shift + Tab para “desfazer o recuo” de uma linha de código. Se quiser fazer recuo em mais de uma linha, selecione todas as linhas que você deseja recuar. Depois, pressione a tecla Tab.

A barra de ferramentas Editar do VBE (que por padrão está oculta) **DICA**

contém dois botões úteis: Recuo e Recuo deslocado. Esses botões possibilitam fazer ou desfazer, rapidamente, recuos de um bloco de código. Selecione o código e clique em um desses botões para mudar o recuo do bloco.

Opção Recurso de edição arrastar e soltar

A opção Recurso de Edição arrastar e soltar, quando habilitada, permite que você copie e mova o texto arrastando e soltando com o seu mouse.

Mantenho esta opção ativada, mas nunca uso. Prefiro copiar e mover usando o teclado.

Opção Padrão para modo de exibição de módulo completo

A opção Padrão para modo de exibição de módulo completo configura a posição padrão para novos módulos (isso não afeta os módulos existentes). Quando habilitada, os procedimentos na janela de código aparecem como uma única lista rolável. Se desabilitada, você pode ver apenas um procedimento por vez. Mantenho essa opção habilitada.

Opção separador de procedimento

Quando a opção Separador de Procedimento está ativada, a barra de separação aparece no final de cada procedimento em uma janela de código. Gosto da ideia de barras de separação, então mantenho essa opção ativada.

Usando a guia Formato do editor

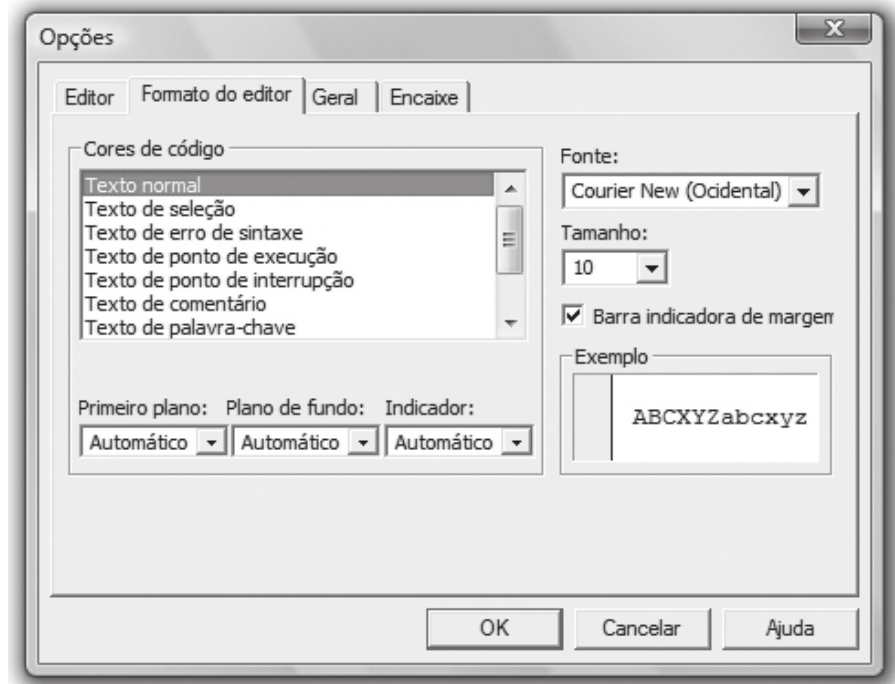
A Figura 3-8 mostra a guia de Formato do Editor da caixa de diálogo Opções. Com essa guia, você pode personalizar a aparência do VBE.

Opção Cores de código

A opção Cores de Código permite que você configure a cor de texto e a cor de fundo exibida em vários elementos do código VBA. Isso é grandemente uma questão de preferência pessoal. Eu, particularmente, acho o padrão de cores bom. Mas, para uma mudança de cenário, ocasionalmente brinco com essas configurações.

ExcelVBABokks.indb 52

15/02/2013 13:59:21



Capítulo 3: Trabalhando no Visual Basic Editor

53

Figura 3-8:

Mude a
aparência
do VBE
com a guia
do editor
Formato.

Opção Fonte

A opção Fonte permite que você selecione a fonte que é usada nos seus módulos VBA. Para melhores resultados, mantenha uma fonte de

largura fixa, como Courier New. Em uma fonte de largura fixa, todos os caracteres têm exatamente a mesma largura. Isso torna seu código mais legível porque os caracteres estão verticalmente alinhados e você pode distinguir facilmente espaços múltiplos (o que, às vezes, é útil).

Configuração de tamanho

A configuração de tamanho especifica a medida do tamanho da fonte nos módulos do VBA. Essa configuração é uma questão de preferência pessoal determinada pela resolução de exibição e de quantas cenouras você tem comido.

Opção de Barra Indicadora de margem

Esta opção controla a exibição da Barra indicadora de margem vertical de seus módulos. Você deve mantê-la ativada; caso contrário, não será capaz de ver os indicadores úteis quando estiver depurando o seu código.

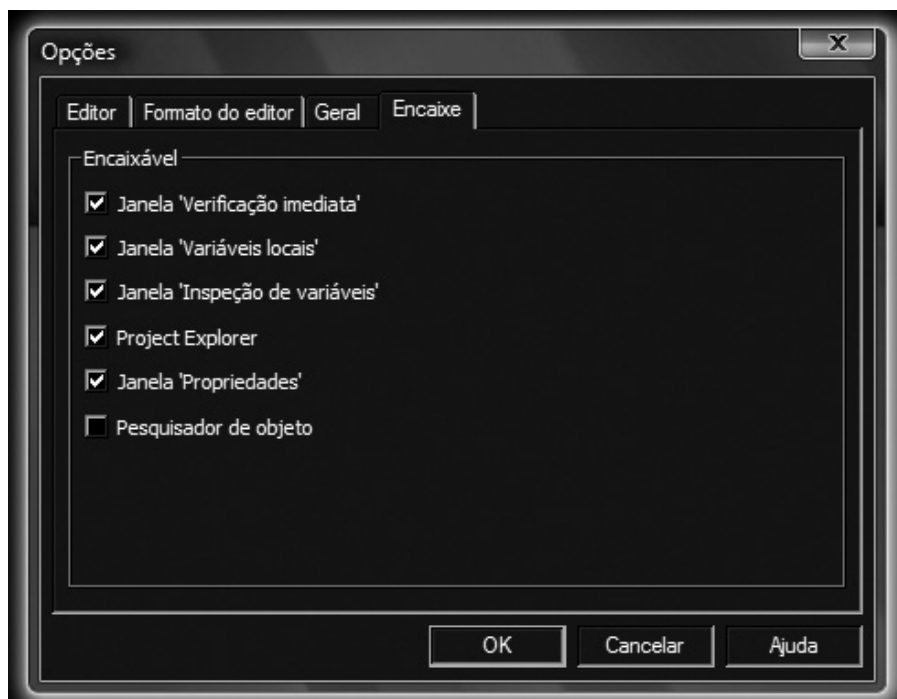
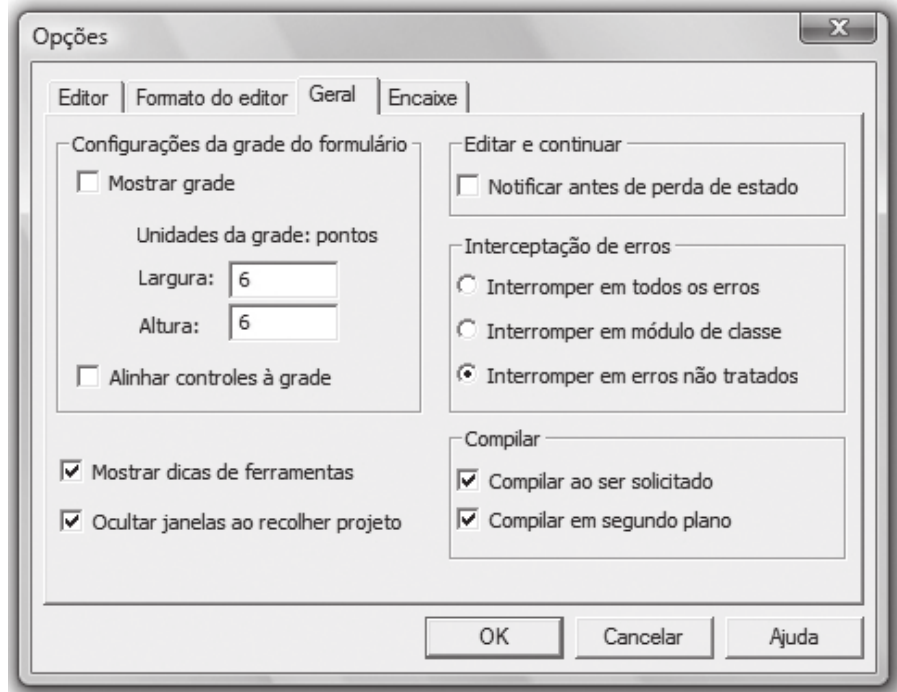
Usando a guia Geral

A Figura 3-9 mostra as opções disponíveis na guia Geral, na caixa de diálogo Opções. Na maioria dos casos, as configurações padrão são boas.

A configuração mais importante é Intercepção de erros. Sugiro com veemência que você habilite a opção Interromper em erros não tratados, que é o padrão. Se usar uma configuração diferente, o tratamento contra erro no código não funciona. Você pode ler mais sobre isso no Capítulo 12.

ExcelVBABokks.indb 53

15/02/2013 13:59:21



54 Parte II: Como o VBA Trabalha com o Excel

Se estiver de fato interessado nestas opções, para detalhes, clique o

botão Ajuda.

Figura 3-9:

A guia Geral

da caixa de

diálogo

Opções.

Usando a guia Encaixe

A Figura 3-10 mostra a guia Encaixe. Essas opções determinam como as diversas janelas no VBE atuam. Quando a janela está encaixada, ela é fixada em um lugar próximo a uma das margens da janela de programa do VBE. Isso facilita identificar e localizar uma janela em especial. Se desativar todos os encaixes, você vai gerar uma grande e confusa bagunça de janelas. Geralmente, as configurações padrão funcionam bem.

Figura 3-10:

A guia

Encaixe da

caixa de

diálogo

Opções.

ExcelVBABokks.indb 54

15/02/2013 13:59:21

Capítulo 4

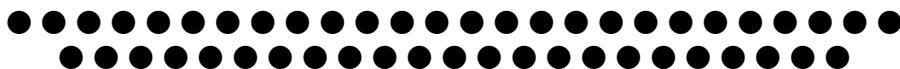
Introdução ao Modelo de

Objeto do Excel



Neste Capítulo

- ☐ Introdução ao conceito de objetos
- ☐ Descobrindo sobre a hierarquia do objeto Excel
- ☐ Entenda as coleções de objeto
- ☐ Como fazer referência a objetos específicos no seu código VBA
- ☐ Como acessar ou mudar as propriedades de um objeto
- ☐ Desempenhando ações com métodos de um objeto



Todos já estão familiarizados com a palavra objeto. Bem, pessoal, esqueça a definição que acha que sabe. No mundo da programa-

ção, objeto tem um significado diferente. Geralmente, você a vê sendo usada como parte da expressão *programação orientada a objeto*, ou resumida como POO. A POO baseia-se na ideia de que software consiste de objetos distintos que têm atributos (ou propriedades) e podem ser manipulados. Esses objetos não são coisas materiais. Ao contrário, eles existem em forma de bits e bytes.

Neste capítulo, eu o apresento ao modelo de objeto do Excel, que é uma hierarquia de objetos contidos no Excel. Quando terminar este capítulo, terá um entendimento bem razoável do que é POO e porque precisa entender esse conceito para se tornar um programador VBA. Afinal, a programação do Excel, na verdade, se resume em manipular objetos do Excel. É simples assim.

O material deste capítulo pode ser um pouco esmagador. Mas, por **BRE-SE**

favor, acate o meu conselho, mesmo se não compreender plenamente **LEM**

na primeira vez. Os conceitos importantes apresentados aqui farão mais sentido à medida que você avançar no livro.

ExcelVBABokks.indb 55

15/02/2013 13:59:21

56 Parte II: Como o VBA Trabalha com o Excel *Excel É um Objeto?*

Você já usou um pouco o Excel, mas provavelmente nunca pensou nele como um objeto. Quanto mais você trabalha com VBA, mais vê o Excel nesses termos. Você entenderá que o Excel é um objeto e que ele contém outros objetos. Esses objetos, por sua vez, contém ainda mais objetos. Em outras palavras, a programação VBA envolve trabalhar com uma hierarquia de objetos.

No alto desta hierarquia está o objeto Application (Aplicativo) — neste caso, o próprio Excel (a mãe de todos os objetos).

Escalando a Hierarquia de Objetos

O objeto Application contém outros objetos. A seguir, é apresentada uma lista de alguns dos objetos mais úteis contidos no Aplicativo Excel: 5 Add-in (suplementos)

5 Window (janela)

5 Workbook (pasta de trabalho)

5 WorksheetFunction (Funções das planilhas)

Cada objeto inserido no objeto Application pode conter outros objetos.

Por exemplo, a seguir está uma lista de objetos que pode estar contida em um objeto Workbook (pasta de trabalho):

5 Chart (Tabela)

5 Name (Nome)

5 VBProject

5 Window (Janela)

5 Worksheet (Planilha)

Por sua vez, cada um desses objetos ainda pode conter mais objetos.

Considere um objeto Worksheet, que está inserido no objeto Workbook, que está inserido no objeto Application. Alguns objetos que podem estar contidos em um objeto Worksheet são:

5 Comment (Comentário)

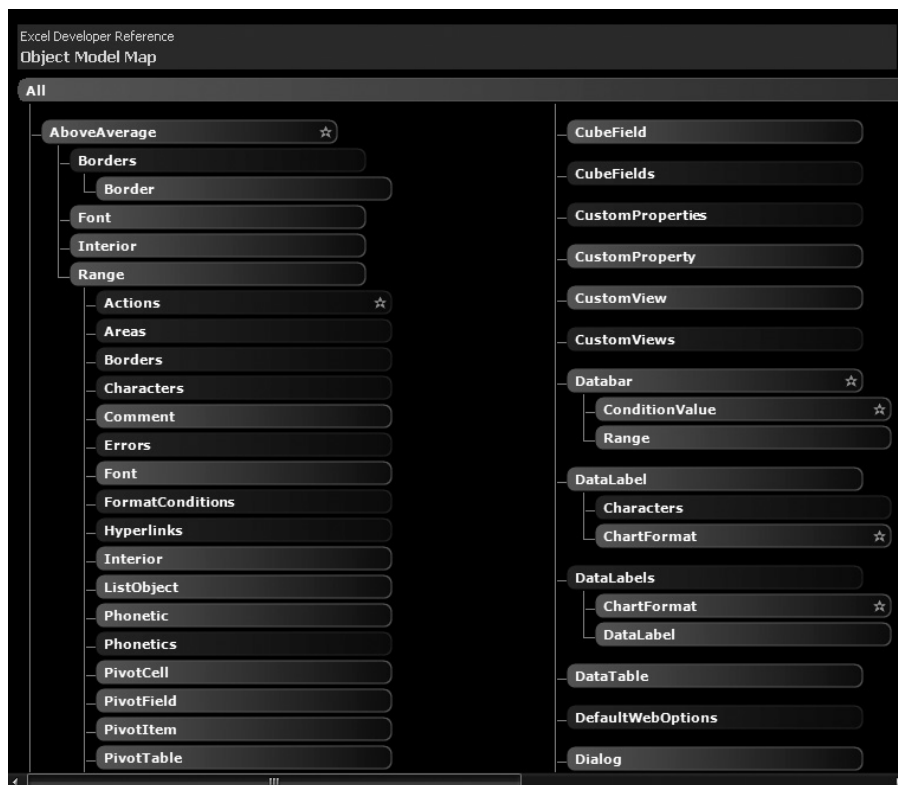
5 Hiperlink

5 Name (Nome)

5 PageSetup (configuração de página)

ExcelVBABokks.indb 56

15/02/2013 13:59:21



Capítulo 4: Introdução ao Modelo de Objeto Excel

57

5 PivotTable (Tabela Principal)

5 Range (Intervalo de Células)

Falando de outra forma, se você quiser fazer algo com um intervalo de células em uma determinada planilha, pode visualizar aquele intervalo da seguinte maneira:

Range⇒contida em Worksheet⇒contida em Workbook⇒contida no Excel

Está começando a fazer sentido?

A Figura 4-1 mostra parte do Mapa de modelo de objeto do Excel. Se você deseja realmente ficar admirado, exiba o sistema de Ajuda VBA e busque pelo mapa de modelo de objeto. Trata-se de um enorme diagrama que relaciona todos os objetos, e cada um é clicável, portanto, você pode ler tudo sobre ele.

Figura 4-1:

Visualizan—

do uma

parte do

modelo

objeto

Excel.

É isso, companheiros, o Excel tem mais objetos do que você pode
BRE-SE

supor e, mesmo sujeitos antigos como eu podem ficar admirados. A
LEM

boa notícia é que você nunca terá que lidar, de fato, com a maioria desses objetos. Quando estiver trabalhando em um problema, basta focalizar alguns objetos relevantes – os quais você geralmente pode descobrir gravando uma macro.

58 Parte II: Como o VBA Trabalha com o Excel *Enchendo Sua Cabeça com Coleções*

As coleções são um outro tipo de conceito-chave em programação VBA.

Uma coleção é um grupo de objetos do mesmo tipo. E, para aumentar a confusão, uma coleção é, ela própria, um objeto.

Aqui estão alguns exemplos das coleções comumente usadas:

5 Pastas de Trabalho: Uma coleção de todos os objetos

Workbook abertos no momento.

5 Planilhas: Uma coleção de todos os objetos de Planilha em um objeto Workbook em especial.

5 Gráficos: Uma coleção de todos os objetos Chart (gráficos) contidos em um objeto Workbook em especial.

5 Planilhas: Uma coleção de todas as planilhas (independente do seu tipo) contidas em um objeto Workbook em especial.

Você deve perceber que aqueles nomes de coleção estão todos no plural, o que faz sentido (pelo menos eu espero).

“Para que servem as coleções?” você deve estar se perguntando. Bem, elas são muito úteis, por exemplo, quando você deseja fazer algo não apenas com uma planilha, mas com várias. Como você verá, o seu código VBA pode fazer loop através de todos os membros de uma coleção, e fazer alguma coisa em cada um deles.

Como Fazer Referência aos Objetos

Apresentei as informações nas seções anteriores para prepará-lo para o próximo conceito: fazer referência aos objetos no seu código VBA.

Fazer referência a um objeto é importante porque você deve identificar o objeto com o qual quer trabalhar. Afinal, o VBA não pode ler sua mente – ainda.

Eu creio que o objeto de leitura de mente será introduzido no Excel 2013.

Você pode trabalhar com uma coleção inteira de objetos de uma só vez.

No entanto, com mais frequência, é preciso trabalhar com um objeto específico em uma coleção (como uma planilha em especial em uma pasta de trabalho). Para fazer referência a um único objeto de uma coleção, você coloca o nome do objeto ou o número do índice entre parênteses depois do nome da coleção, assim: Worksheets(“Plan1”)

Note que o nome da planilha está entre aspas. Se omitir as aspas, o Excel não conseguirá identificar o objeto (o Excel pensará que é um nome de variável).

Se a Plan1 é a primeira (ou a única) planilha na coleção, você também pode usar a seguinte referência:

ExcelVBABokks.indb 58

15/02/2013 13:59:22

Capítulo 4: Introdução ao Modelo de Objeto Excel

59

Worksheets(1)

Neste caso, o número não está entre aspas. Qual é a questão? Se você **BRE-SE**

faz referência a um objeto usando seu nome, use as aspas. Se referen-
LEM

ciar um objeto usando seu número de índice, use um número inteiro sem aspas.

Outra coleção, chamada Sheets (Planilhas), contém todas as planilhas (planilhas e gráficos) em uma pasta de trabalho. Se a Plan1 for a primeira planilha na pasta de trabalho, você pode referenciá-la como

Se você quer trabalhar com o objeto Application, é fácil: comece digitando **Application**.

Todos os outros objetos no modelo objeto do Excel estão sob o objeto Application. Você obtém esses objetos descendo na hierarquia e conectando cada objeto à sua maneira, com o operador ponto (.). Para ter o objeto Workbook chamado “Pasta1.xlsx”. comece com o objeto Application e navegue para o objeto da coleção Workbooks.

Application.Workbooks(“Pasta1.xlsx”)

Para navegar além de uma planilha específica, adicione um operador ponto e acesse o objeto da coleção Worksheets.

Application.Workbooks(“Pasta1.xlsx”).Worksheets(1)

Ainda não é suficiente? Se quiser obter o valor da célula A1 na primeira planilha da pasta de trabalho chamada Pasta1.xlsx, é preciso passar para o nível do objeto Range.

Application.Workbooks(“Book1.xlsx”). _

Worksheets(1).Range(“A1”).Value

Ao fazer referência a um objeto Range dessa forma, isso é chamado de referência totalmente qualificada. Você informou ao Excel exatamente qual intervalo deseja, em qual planilha e pasta de trabalho, e não deixou qualquer coisa à imaginação. A imaginação é boa nas pessoas, mas não em programas de computador.

A propósito, nomes de pasta de trabalho também têm um ponto para separar o nome de arquivo da extensão (por exemplo, Pasta1.xlsx). Isso é apenas uma coincidência. O ponto em um nome de arquivo nada tem a ver com o operador ponto ao qual me referi há alguns parágrafos.

ExcelVBABokks.indb 59

15/02/2013 13:59:22

60 Parte II: Como o VBA Trabalha com o Excel Simplificando referências a objetos

Se você teve que qualificar toda referência ao objeto que fez, seu código deve estar um pouco longo e deve estar mais difícil de ler.

Felizmente, o Excel fornece alguns atalhos que podem melhorar a legibilidade (e poupá-lo de alguma digitação). Para os iniciantes, o objeto Application é sempre hipotético. Existem apenas alguns casos em que faz sentido digitá-lo. Omitir a referência ao objeto Application encurta o exemplo da seção anterior para:

```
Workbooks("Pasta1.xlsx").Worksheets(1).Range("A1").Value
```

Este é um bom aperfeiçoamento. Mas, espere, tem mais. Se você tiver certeza de que Pasta1.xlsx é a pasta de trabalho ativa, também pode omitir tal referência. Agora, ficamos com `Worksheets(1).Range("A1").Value`

Agora estamos chegando em algum lugar. Você adivinhou o próximo atalho? Está certo, se a primeira planilha é a planilha ativa atualmente, então o Excel aceitará aquela referência e nos permitirá apenas digitar: `Range("A1").Value` **BRE-SE**

Ao contrário do que muitas pessoas pensam, o Excel não tem um objeto **LEM**

Cell (célula). Uma célula é simplesmente um objeto Range (faixa) que consiste apenas de um elemento.

Os atalhos descritos aqui são bons, mas também podem ser perigosos.

E se você apenas pensar que o Pasta1.xlsx é a pasta de trabalho ativa?

Você pode obter um erro, ou pior, o valor errado e nem perceber que está errado. Por essa razão, muitas vezes é melhor qualificar totalmente suas referências aos objetos.

No Capítulo 14, discuto a estrutura With-End With, que ajuda a qualificar totalmente suas referências, mas também ajuda a tornar o código mais legível e diminui a digitação. O melhor dos dois mundos!

Mergulhando nas Propriedades e nos

Métodos do Objeto

Embora saber como referir aos objetos seja importante, você não pode fazer nada de útil simplesmente referindo a um objeto (como nos exemplos das seções anteriores). Para realizar algo significativo, você deve fazer uma dessas duas coisas:

5 Ler ou modificar as propriedades de um objeto.

5 Especificar um método de ação para ser usado com um objeto.

ExcelVBABokks.indb 60

15/02/2013 13:59:22

Capítulo 4: Introdução ao Modelo de Objeto Excel

61

Com milhares de propriedades e métodos disponíveis, literalmente, é possível ficar confuso com facilidade. Eu tenho trabalhado há anos com essa coisa e ainda estou confuso. Mas, como eu disse antes e repito: você nunca precisará usar a maioria das propriedades e métodos disponíveis.

DE ESPECIALISTA

O P A P

Uma outra perspectiva sobre

McObjects, McProperties e McMethods

Eis uma analogia que pode ajudá-lo a entender o Até agora, tudo bem. Essa analogia parece fun-relacionamento entre objetos, propriedades e cionar. Vejamos se posso levá-la adiante.

métodos em VBA. Nessa analogia, eu comparo Os objetos do Excel têm propriedades. Por Excel a uma cadeia de lanchonetes.

exemplo, um objeto Range tem propriedades

A unidade básica de Excel é um objeto tais como Value (valor) e Name (nome) e um ob-Workbook (pasta de trabalho). Em uma cadeia jeto Shape (forma) tem propriedades como Wi-de lanchonetes, a unidade básica está em um dth (largura), Height (altura) e assim por diante.

restaurante individual. Com o Excel, você pode Sem surpresas, os objetos em uma lanchonete acrescentar uma pasta de trabalho e fechá-la, também têm propriedades. O objeto Fogão, por e todas as pastas de trabalho abertas são exemplo, tem propriedades como Temperatura conhecidas como Workbook (uma coleção de e Números de Bocas. O Ventilador tem o seu objetos Workbook). Da mesma forma, a próprio conjunto de propriedades (Ligado, Rota-administração de uma cadeia

de lanchonetes cão Por Minuto etc.).

pode acrescentar um restaurante e fechar um Além das propriedades, os objetos do Excel restaurante, e todos os restaurantes da cadeia também têm métodos, os quais executam uma podem ser vistos como a coleção Restaurantes operação em um objeto. Por exemplo, o método (uma coleção de objetos Restaurante).

do ClearContents (limpar conteúdo) apaga o

Uma pasta de trabalho do Excel é um objeto, conteúdo de um objeto Range. Um objeto em mas também contém outros objetos, tais como uma lanchonete também tem métodos. É pos-planilhas, gráficos, módulos VBA e assim por sível prever com facilidade um método Terdiante. Além do mais, cada objeto em uma mostato em um objeto Fogão ou um método pasta de trabalho pode conter seus próprios Ligar em um objeto Ventilador.

objetos. Por exemplo, um objeto Worksheet Às vezes, em Excel, os métodos mudam as pro-

(planilha) pode conter objetos Range (faixa), priedades de um objeto. O método ClearContents objetos PivotTable (tabela principal), objetos em uma Range muda a propriedade Valor da Shape e assim por diante.

Range. Da mesma forma, o método Termostato

Continuando com a analogia, uma lanchonete em um objeto Fogão afeta a sua propriedade (como uma pasta de trabalho) contém objetos Temperatura. Com VBA, você pode escrever pro-tais como a Cozinha, a Área de Refeições e Me-cedimentos para manipular objetos do Excel. Em sas (uma coleção). Além disso, a administração uma lanchonete, a administração pode dar or-pode acrescentar ou remover objetos do objeto dens para manipular os objetos nos restauran-Restaurante. Por exemplo, a administração pode tes. (“Acenda o fogão e acelere o ventilador”).

adicionar mais mesas à coleção Mesas. Cada Da próxima vez em que você visitar a sua lan-um desses objetos pode conter outros objetos. chonete preferida, diga apenas “Eu quero um Por exemplo, o objeto Cozinha tem um objeto Fo-objeto Burger com a propriedade Cebola configuração, um objeto Ventilador, um objeto Chefe de gurada para Falso”.

cozinha, um objeto Pia) e daí por diante.

ExcelVBABokks.indb 61

15/02/2013 13:59:22



62 Parte II: Como o VBA Trabalha com o Excel

Propriedades do objeto

Todo objeto tem propriedades. Você pode pensar em propriedades como atributos que descrevem o objeto. A propriedade de um objeto determina sua aparência, como ele age e até se ele é visível. Usando o VBA, você pode fazer duas coisas com as propriedades de um objeto: 5 Examinar a configuração atual de uma propriedade.

5 Mudar a configuração da propriedade.

Por exemplo, um objeto Range de uma única célula tem uma propriedade chamada Value. A propriedade Value armazena o valor contido na célula.

Você pode escrever o código VBA para exibir a propriedade Value, ou pode escrever o código VBA para configurar um valor específico à propriedade Value. A macro a seguir usa a função MsgBox integrada em VBA para apresentar uma caixa que exibe o valor na célula A1 na Planilha1 da pasta de trabalho ativa. Veja a Figura 4-2.

```
Sub ShowValue()
```

```
Contents = Worksheets("Plan1").Range("A1").Value
```

```
MsgBox Contents
```

```
End Sub
```

Figura 4-2:

Esta caixa de

mensagem

mostra a

propriedade

Value de um

objeto Range.

A propósito, a MsgBox é uma função útil. Você pode usá-la para exibir resultados enquanto o Excel executa o seu código VBA. Direi mais sobre essas funções no Capítulo 15, portanto, tenha paciência (ou simplesmente pule e leia tudo sobre ela).

O código no exemplo anterior mostra a configuração atual da propriedade Valor de uma célula. E se você quiser mudar a configuração para aquela propriedade? A macro seguinte muda o valor exibido na célula A1 alterando a propriedade Value da célula:

```
Sub ChangeValue()
```

```
Worksheets("Plan1").Range("A1").Value = 94,92
```

```
End Sub
```

ExcelVBABokks.indb 62

15/02/2013 13:59:22

Capítulo 4: Introdução ao Modelo de Objeto Excel

63

Depois que o Excel executa esse procedimento, a célula A1 na Planilha1

da pasta de trabalho ativa passa a conter o valor de 94,92. Se a pasta de trabalho ativa não tiver uma planilha nomeada Plan1, ao executar a macro será exibida uma mensagem de erro. O VBA apenas segue instruções e não pode trabalhar com uma planilha que não existe.

Cada objeto tem seu próprio conjunto de propriedades, embora algumas propriedades sejam comuns a todos os objetos. Por exemplo, muitos (mas não todos) objetos têm uma propriedade Visible (visível).

A maioria dos objetos também tem uma propriedade Name.

Algumas propriedades de objeto são apenas para leitura, o que significa que você pode ver o valor da propriedade, mas não pode mudá-lo.

Como mencionei anteriormente neste capítulo, uma coleção também é **BRE-SE**

um objeto. Isto significa que uma coleção também tem propriedades. Por **LEM**

exemplo, você pode determinar como várias pastas de trabalho são abertas acessando a propriedade Count da coleção Workbooks. O

seguinte procedimento VBA exibe uma caixa de mensagem informando quantas pastas de trabalho estão abertas:

```
Sub CountBooks()
```

```
MsgBox Workbooks.Count
```

```
End Sub
```

Métodos de Objeto

Além das propriedades, os objetos têm métodos. Um método é uma ação que você executa com um objeto. Um método pode mudar propriedades de um objeto ou fazer com que o objeto faça alguma coisa.

Este simples exemplo usa o método ClearContents em um objeto Range para apagar o conteúdo da célula A1 na planilha ativa:

```
Sub ClearRange()
```

```
Range("A1").ClearContents
```

End Sub

Alguns métodos tomam um ou mais argumentos. Um argumento é um valor que especifica mais a ação a ser executada. Você coloca os argumentos para um método depois dele, separado por um espaço.

Argumentos múltiplos são separados por vírgula.

O exemplo a seguir ativa Plan1 (na pasta de trabalho ativa) e depois copia o conteúdo da célula A1 para a célula B1 usando o método Copiar do objeto Range. Nesse exemplo, o método Copy tem um argumento, o intervalo (range) destinado à operação de cópia: Sub CopyOne()

```
Worksheets("Plan1").Activate
```

```
Range("A1").Copy Range("B1")
```

End Sub

ExcelVBABokks.indb 63

15/02/2013 13:59:22

64 Parte II: Como o VBA Trabalha com o Excel Perceba que omiti a referência à planilha quando refiro aos objetos Range. Eu posso fazer isso com segurança, pois usei uma declaração para ativar Plan1 (usando o método Activate).

BRE-SE

Por uma coleção também ser um objeto, as coleções têm métodos. A **LEM**

seguinte macro usa o método Add para a coleção Workbooks:

```
Sub AddAWorkbook()
```

```
Workbooks.Add
```

End Sub

Conforme esperado, esta declaração cria uma nova pasta de trabalho.

Em outras palavras, ela acrescenta uma nova pasta de trabalho à coleção Workbooks.

Eventos de objeto

Nesta seção, citei rapidamente em mais de um tópico que você precisa saber a respeito de eventos. Objetos respondem a vários eventos que ocorrem. Por exemplo, quando você estiver trabalhando no Excel e ativar uma pasta de trabalho diferente, acontece um evento *Activate*.

Você poderia, por exemplo, designar uma macro VBA para ser executada sempre que houver um evento *Activate* de um objeto *Workbook*, em especial.

O Excel suporta vários eventos, mas nem todos os objetos podem responder a todos os eventos. E alguns objetos não respondem a nenhum evento. Os únicos eventos que você pode usar são aqueles disponibilizados pelos programadores da Microsoft Excel. O conceito de um evento se torna claro no Capítulo 11 e também na parte IV.

Descobrindo Mais

Pense em si mesmo como um iniciado no maravilhoso mundo de

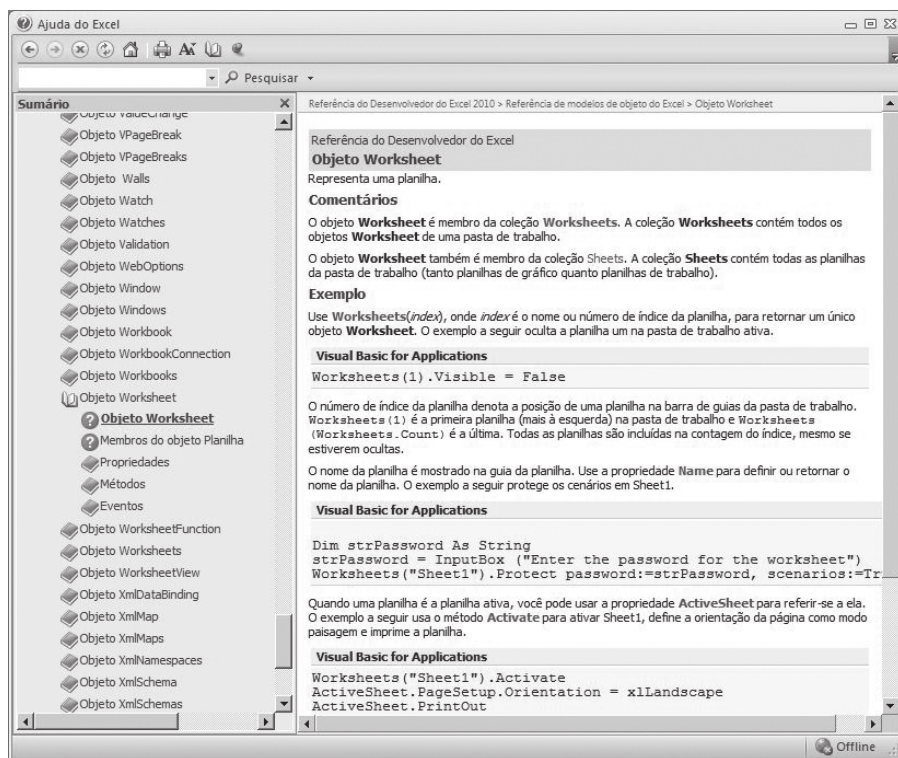
objetos, propriedades, métodos e eventos. Você descobre mais sobre esses conceitos nos próximos capítulos. Se não foi o suficiente, você também pode se interessar por três outras excelentes ferramentas: 5 Sistema de ajuda do VBA

5 O navegador do objeto

5 Lista automática de membros

ExcelVBABokks.indb 64

15/02/2013 13:59:22



Capítulo 4: Introdução ao Modelo de Objeto Excel

65

Usando o sistema de Ajuda de VBA

O sistema de ajuda do VBA descreve cada objeto, propriedade e método disponível para você. Este é um excelente recurso para descobrir mais sobre VBA e é mais compreensível que qualquer livro no mercado.

Se você estiver trabalhando em um módulo VBA e quiser informações

DICA

sobre um objeto, método ou propriedade especial, mova o cursor para a palavra na qual está interessado e pressione F1. Em alguns segundos, você verá o tópico apropriado, completo com referências cruzadas e talvez até um ou dois exemplos.

A Figura 4-3 mostra uma tela do sistema de Ajuda online – neste caso, para um objeto Planilha.

5 Clique em Propriedades para obter uma lista completa de propriedades deste objeto.

5 Clique em Métodos para obter a listagem dos seus métodos.

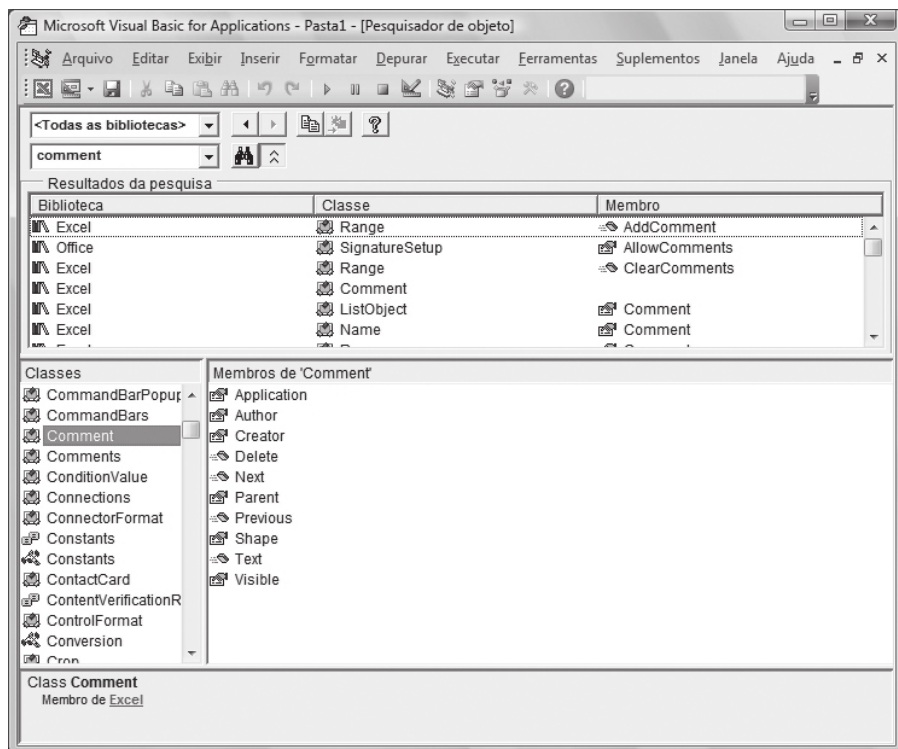
5 Clique em Eventos para obter uma listagem dos eventos aos quais ele responde.

Figura 4-3:

Um exemplo
do sistema
de Ajuda do
VBA.

ExcelVBABokks.indb 65

15/02/2013 13:59:22



66 Parte II: Como o VBA Trabalha com o Excel

Como usar o Pesquisador de Objeto

O VBE possui outra ferramenta, conhecida como Pesquisador de

Objeto. Como o nome sugere, essa ferramenta permite pesquisar os objetos disponíveis. Para acessar o Pesquisador de Objeto, pressione F2 quando o VBE estiver ativo (ou selecione Exibir⇒Pesquisador de objeto). Você verá uma janela como a mostrada na Figura 4-4.

Figura 4-4:

Navegando pelos

objetos

com o

Navegador

de Objeto.

A lista drop-down no alto da janela apresenta uma relação de todas as

bibliotecas de objeto disponíveis no momento. A Figura 4-4 mostra a opção Todas as Bibliotecas. Se você quiser navegar pelos objetos do Excel, selecione Excel na lista drop-down.

A segunda lista drop-down é onde você insere uma busca de string. Por exemplo, se quiser ver todos os objetos do Excel que lidam com comentários, digite **comment** no segundo campo e clique no botão Pesquisar (ele é um binóculo). A janela com os resultados da pesquisa mostra tudo que estiver no objeto biblioteca contendo o texto comentário. Se você ver algo parecido que deve ser interessante, selecione e pressione F1

para mais informações.

Como relacionar automaticamente

propriedades e métodos

No Capítulo 3, mencionei um punhado de itens chamados de Membros de autolistagem (conhecidos como “IntelliSense”). Esse recurso oferece uma lista de propriedades e métodos enquanto você digita. A Figura 4-5

mostra um exemplo da coleção Workbooks.

ExcelVBABokks.indb 66

15/02/2013 13:59:22

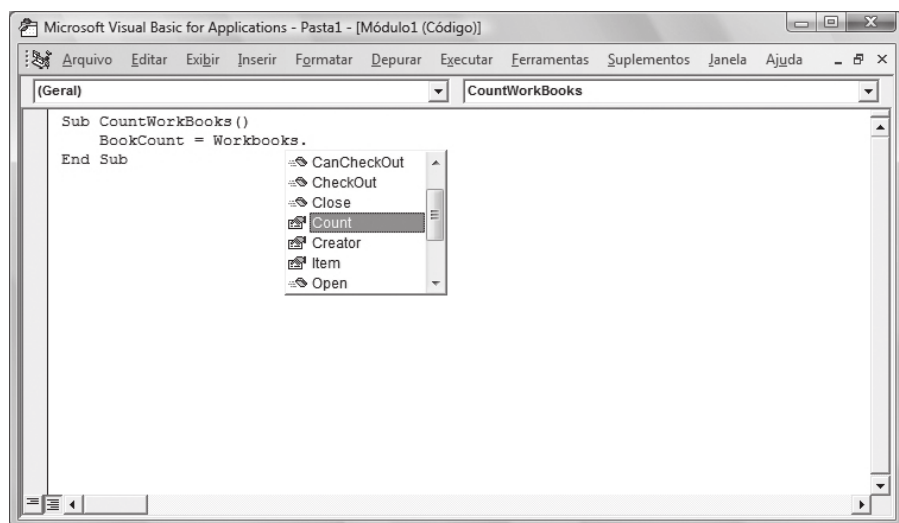


Figura 4-5: O

recurso

Autolistar

Membros o

ajuda a

identificar

propriedades

e métodos

para um

objeto.

Depois que eu digitei o ponto, após Workbooks, o VBE se ofereceu para ajudar a exibir uma lista de propriedades e métodos para aquela coleção. Depois que digitei o c, a listagem se restringiu aos itens que começam com essa letra. Selecione o item que você precisa, pressione Tab e pronto! Você poupou um pouco de digitação — e ainda garantiu que a propriedade ou o método foram corretamente digitados.

ExcelVBABokks.indb 67

15/02/2013 13:59:23

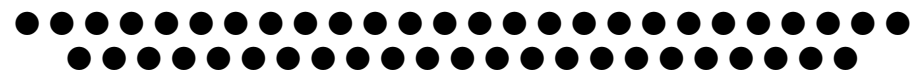
68 Parte II: Como o VBA Trabalha com o Excel ExcelVBABokks.indb 68

15/02/2013 13:59:23

Capítulo 5

Procedimentos Function e

Sub no VBA



Neste Capítulo

- ☐ Entendendo a diferença entre procedimentos Sub e procedimentos Function
- ☐ Executando procedimentos Sub (várias formas)
- ☐ Executando procedimentos Function (duas formas)



Diversas vezes nos capítulos anteriores, mencionei os *procedimentos Sub* e me referi ao fato de que procedimentos Function também têm uma função em VBA. Neste capítulo, esclareço a confusão desses conceitos.

Subs versus Funções

O código VBA que você escreve no VBE é conhecido como um *procedimento*. Os dois tipos de procedimentos mais comuns são Sub e Function.

5 Um procedimento Sub é um grupo de declarações de VBA que executam uma ação (ou várias ações) no Excel.

5 Um procedimento Function é um grupo de declarações que executa um cálculo e retorna um único valor.

A maioria das macros que você escreve no VBA são procedimentos Sub.

Você pode pensar que um procedimento Sub é como um comando:

Execute o procedimento Sub e algo acontece (é claro, o que acontece exatamente depende do código VBA do procedimento Sub).

Uma função também é um procedimento, mas é bem diferente de um Sub. O conceito de uma função já é familiar para você. O Excel envolve muitas funções de planilha que você usa diariamente (bem, pelo menos durante a semana). Exemplos incluem SUM, PMT e

VLOOKUP. Você usa essas funções de planilhas em fórmulas. Cada função tem um ou mais argumentos (embora algumas funções não usem argumento algum). A função faz alguns cálculos ocultos e retorna um valor único. O mesmo serve para os procedimentos Function que você desenvolve com VBA.

ExcelVBABokks.indb 69

15/02/2013 13:59:23

70 Parte II: Como o VBA Trabalha com o Excel *Observando os procedimentos Sub*

Todo procedimento Sub inicia com a palavra-chave Sub e termina com uma declaração End Sub. Eis um exemplo:

```
Sub ShowMessage()
```

```
MsgBox "That's all folks!"
```

```
End Sub
```

Esse exemplo mostra um procedimento chamado ShowMessage. O

nome do procedimento antes dos parênteses. Na maioria dos casos, esses parênteses estão vazios. Entretanto, você pode passar argumentos aos procedimentos Sub a partir de outros procedimentos. Se o seu Sub usa argumentos, liste-os entre parênteses.

Quando você grava uma macro no Excel, o resultado é sempre um **BRE-SE**

procedimento Sub.

LEM

Como será visto mais adiante neste capítulo, o Excel fornece várias maneiras para executar um procedimento Sub do VBA.

Observando os procedimentos Function

Todo procedimento Function começa com a palavra-chave Function e termina com uma declaração End Function. Aqui está um simples exemplo: Function CubeRoot(number)

CubeRoot = number ^ (1 / 3)

End Function

Essa função, chamada CubeRoot (raiz cúbica), tem um argumento (chamado *number*), que está entre parênteses. As funções podem ter qualquer quantidade de argumentos ou nenhum. Quando você executa a Function, ela retorna um valor único – a raiz cúbica do argumento passado para a função.

O VBA permite que você especifique qual tipo de informação (também **DICA**

conhecido como tipo de dados) é retornado por um procedimento Function. O Capítulo 7 contém mais informações sobre a especificação de tipos de dados.

Só é possível executar um procedimento Function de duas maneiras.

Você pode executá-lo de outro procedimento (um Sub ou outro procedimento Function) ou usá-lo em uma fórmula de planilha.

BRE-SE

Independente de quanto você tente, não é possível usar o gravador de macro do Excel para gravar um procedimento Function. Você precisa **LEM**

inserir manualmente todo procedimento Function que criar.

ExcelVBABokks.indb 70

15/02/2013 13:59:23

Capítulo 5: Procedimentos Function e Sub no VBA

71

Nomeando Subs e Functions

Como os humanos, os animais de estimação e os furacões, todo procedimento Sub e Function deve ter um nome. Embora seja perfeitamente aceitável nomear seu cachorro de Hairball Harris, geralmente essa não é uma boa ideia para nomear procedimentos. Quando nomear procedimentos, você deve seguir algumas regras:

5 Você pode usar letras, números e alguns caracteres de

pontuação, mas o primeiro caractere precisa ser uma letra.

5 Você não pode usar quaisquer espaços ou pontos no nome.

5 O VBA não distingue entre letras maiúsculas e letras minúsculas.

5 Você não pode colocar nenhum dos caracteres a seguir em um nome: #, \$, %, @, ^, * ou !.

5 Se você escrever um procedimento de Function para usar em uma fórmula, não use um nome que se pareça com um endereço de célula (por exemplo, AK47). Na verdade, o Excel permite tais nomes de função, mas por que tornar as coisas mais confusas do que já são?

5 Os nomes de procedimentos não podem ser maiores que 255 caracteres (é claro, você nunca faria um nome de procedimento tão longo).

De maneira ideal, o nome de um procedimento descreve o objetivo de uma rotina. Um bom exercício é criar um nome associando a um verbo e a um substantivo — por exemplo, `ProcessarDados`, `ImprimirRelatório`, `Classificar_Array` ou `VerificarNomeDeArquivo`.

Alguns programadores preferem usar nomes como sentença que oferecem uma descrição completa do procedimento. Alguns exemplos incluem `EscreverRelatórioParaArquivodeTexto` e `Imprimir_Opções_e_`

`Imprimir_Relatório`. O uso de nomes tão longos tem prós e contras. Por um lado, tais nomes são descritivos e, normalmente, sem

ambigüidade.

Por outro lado, demoram mais tempo para digitar. Cada pessoa desenvolve um estilo de nomeação, mas os principais objetivos são aqueles de fazer nomes descritivos e evitar nomes sem sentido, como Pateta, Atualizar, Corrigir e Macro1.

Executando Procedimentos Sub

Embora você possa não saber muito sobre desenvolvimento de procedimentos Sub nesse momento, vou me adiantar um pouco e discutir como executar esses procedimentos. Isso é importante, porque um procedimento Sub é inútil a menos que você saiba como executá-lo.

ExcelVBABokks.indb 71

15/02/2013 13:59:23

72 Parte II: Como o VBA Trabalha com o Excel A propósito, *executar* um procedimento Sub significa a mesma coisa que fazer funcionar ou *chamar* um procedimento Sub. É possível usar qualquer terminologia que você queira.

Como eu o executo? Deixe-me contar as maneiras. Você pode executar um VBA Sub de muitas formas — essa é uma razão pela qual você pode fazer tantas coisas úteis com procedimentos Sub. Eis uma lista exaustiva das maneiras (bem, pelo menos todas as maneiras que pude lembrar) para executar um procedimento Sub:

5 Com o comando Executar⇒Executar Sub/UserForm (no VBE). O

Excel executa o procedimento Sub no qual o cursor está

localizado. Este comando de menu tem duas alternativas: a

tecla F5 e o botão Executar/UserForm na barra de ferramentas

Padrão do VBE. Esses métodos não funcionam se o procedimento

exigir um ou mais argumentos.

5 Da caixa de diálogo Macro do Excel. Você abre essa caixa

selecionando Desenvolver⇒Código⇒Macros ou escolhendo

Exibição⇒Macros⇒Exibir Macros. Ou deixar de lado as guias e

simplesmente pressionar a tecla de atalho Alt + F8. Quando a

caixa de diálogo Macro aparecer, selecione o procedimento Sub que deseja e clique Executar. Essa caixa de diálogo lista apenas os procedimentos que não exigem um argumento.

5 Usando a tecla de atalho Ctrl + Tecla designada ao procedimento Sub (supondo que você designou uma).

5 Clicando em um botão ou em uma forma na planilha. O botão ou a forma devem ter um procedimento Sub designado a ele.

5 A partir de outro procedimento Sub que você escreve.

5 De um botão que você acrescentou à barra de ferramentas de Acesso Rápido (veja o Capítulo 19).

5 A partir de um item personalizado na lista que você desenvolveu (veja o Capítulo 19).

5 Automaticamente, quando você abrir ou fechar uma pasta de trabalho (veja o Capítulo 11).

5 Quando ocorrer um evento. Como explico no Capítulo 11, esses eventos incluem salvar a pasta de trabalho, fazer uma alteração em uma célula, ativar uma planilha e outras coisas.

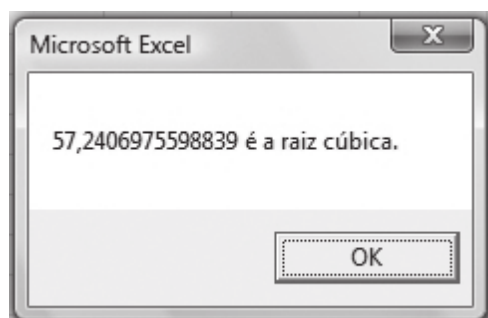
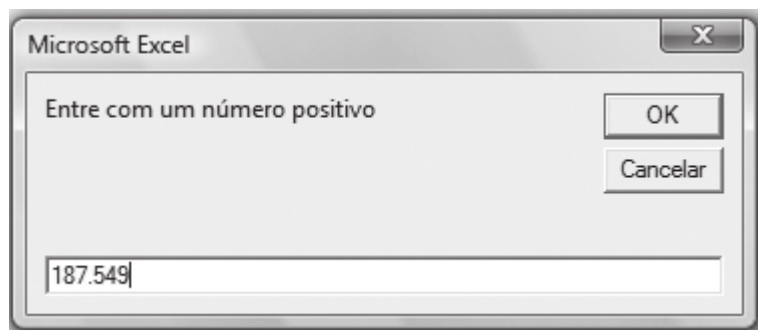
5 Da janela Verificação Imediata no VBE. Apenas digite o nome do procedimento Sub e pressione Enter.

Demonstrei algumas dessas técnicas nas seções seguintes. Antes de continuar, você precisa inserir um procedimento Sub em um módulo VBA.

1. Comece com uma nova pasta de trabalho.
2. Pressione Alt + F11 para ativar o VBE.
3. Selecione a pasta de trabalho na janela de Projeto.

ExcelVBABokks.indb 72

15/02/2013 13:59:23



Capítulo 5: Procedimentos Function e Sub no VBA

73

4. Selecione Inserir⇒Módulo para inserir um novo módulo.
5. Insira o seguinte no módulo:

```
Sub Raiz Cúbica()
```

```
Num = InputBox("Entre com um número positivo")
```

```
MsgBox Num ^ (1/3) & " é a raiz cúbica."
```

```
End Sub
```

Este procedimento pede ao usuário por um número e depois exibe a raiz cúbica daquele número em uma caixa de mensagem. As Figuras 5-1

e 5-2 mostram o que acontece quando você executa este procedimento.

Figura 5-1:

Usando a

função

interna do

VBA

InputBox

para obter

um número.

Figura 5-2:

Exibindo a

raiz cúbica

de um

número

através da

função

MsgBox

A propósito, RaizCúbica não é um exemplo de uma *boa* macro. Ela não

verifica erros, então falha facilmente. Para ver o que quero dizer, tente clicar no botão Cancel (Cancelar) na caixa de entrada ou insira um número negativo.

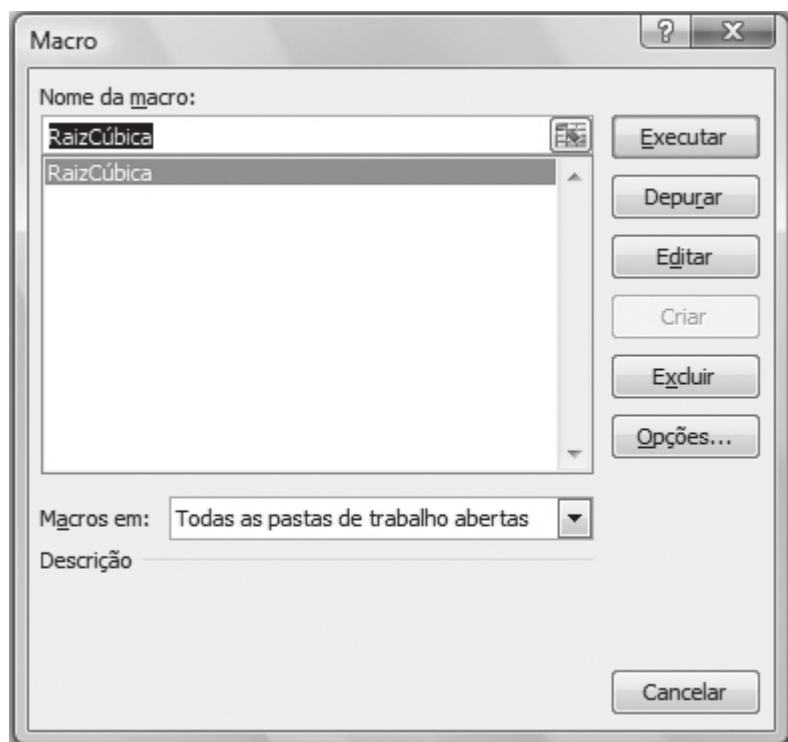
Executando diretamente o procedimento Sub

A maneira mais rápida de executar esse procedimento é ir diretamente ao módulo VBA no qual você o definiu. Siga esses passos:

1. Ative o VBE e selecione o módulo VBA que contém o procedimento.

ExcelVBABokks.indb 73

15/02/2013 13:59:23



74 Parte II: Como o VBA Trabalha com o Excel

2. Mova o cursor para qualquer lugar no código do procedimento.

3. Pressione F5 (ou selecione Executar⇒Executar Sub/UserForm).

4. Responda à caixa de entrada e clique OK.

O procedimento exibe a raiz cúbica do número inserido.

Você não pode usar o comando Executar⇒Executar Sub/UserForm para **BRE-SE**

executar um procedimento Sub que usa argumentos, pois não há como **LEM**

passar os argumentos para o procedimento. Se o procedimento contiver um ou mais argumentos, a única forma de executá-lo é chamá-lo a partir de outro procedimento – o qual deve fornecer o(s) argumento(s).

Executando um procedimento a partir da

caixa de diálogo Macro

Na maior parte do tempo, você executa os procedimentos Sub do Excel, não do VBE. Os passos a seguir descrevem como executar uma macro, usando a caixa de diálogo Macro do Excel: **1. Ative o Excel.**

Alt + F11 é a estrada expressa.

2. Selecione Desenvolvedor⇒Código⇒Macros (ou pressione Alt + F8).

O Excel exibe a caixa de diálogo mostrada na Figura 5-3.

3. Selecione a macro.

4. Clique em Executar (ou clique duas vezes no nome da macro na caixa de lista).

Figura 5-3: A

caixa de

diálogo

Macro

relaciona

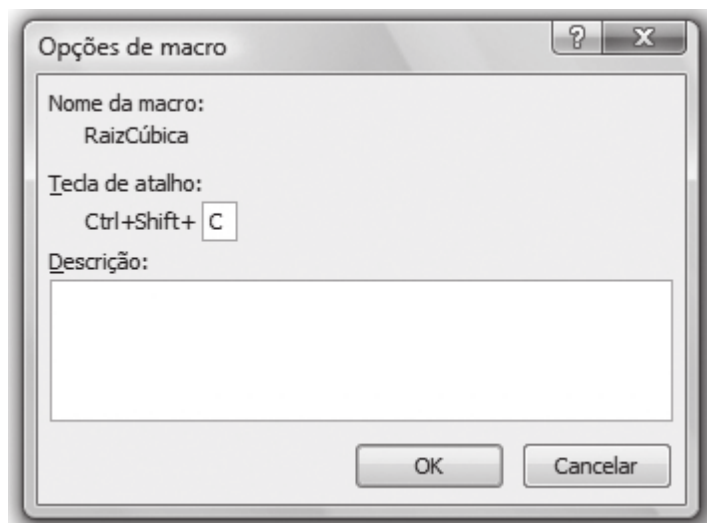
todos os

procedimentos Sub

disponíveis.

ExcelVBABokks.indb 74

15/02/2013 13:59:23



Capítulo 5: Procedimentos Function e Sub no VBA

75

Executando uma macro usando

uma tecla atalho

Uma outra forma de executar uma macro é pressionar a sua tecla de atalho. Mas, antes de poder usar esse método, você precisa atribuir uma tecla de atalho para a macro.

Você tem a oportunidade de atribuir uma tecla de atalho na caixa de diálogo Gravar Macro quando começar a gravar uma macro. Se você criar o procedimento sem usar o gravador de macro, pode atribuir uma tecla de atalho (ou mudar uma tecla de atalho existente) usando o seguinte procedimento: **1. Selecione Desenvolvedor⇒Código⇒Macros.**

2. Selecione o nome do procedimento Sub na caixa de listagem.

Nesse exemplo, o nome do procedimento é RaizCúbica.

3. Clique no botão Opções.

O Excel exibe a caixa de diálogo mostrada na Figura 5-4.

4. Clique a opção Tecla de Atalho e insira uma letra na caixa rotulada Ctrl.

A letra inserida corresponde à combinação de teclas que você

quer usar para executar a macro. Por exemplo, se inserir a letra minúscula c, você pode executar a macro pressionando Ctrl + C. Se você entrar com uma letra maiúscula, será preciso adicionar a tecla Shift à combinação de teclas. Por exemplo, se entrar com C, você pode executar a macro pressionando Ctrl + Shift + C.

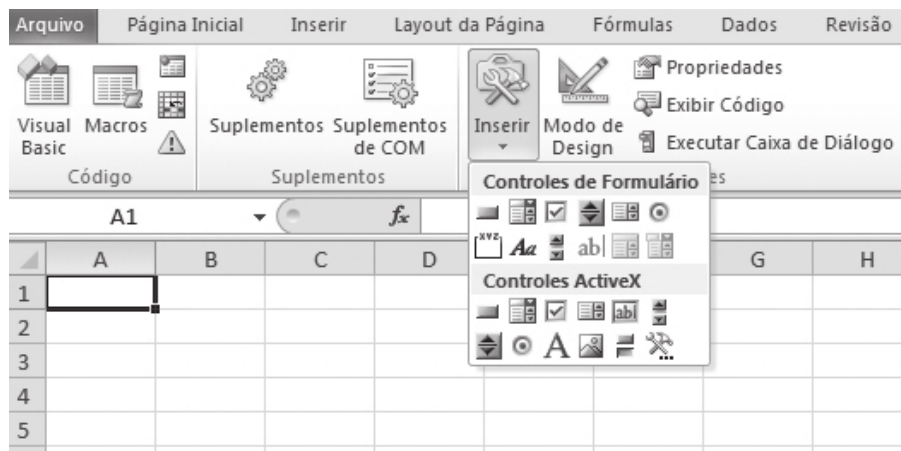
5. Clique OK ou Cancelar para fechar a caixa de diálogo

Opções de Macro.

Figura 5-4:

A caixa de
diálogo
Macro
Options
permite que
você
configure
opções às
suas macros.

Depois de ter atribuído uma tecla de atalho, você pode pressionar aquela combinação de teclas que definiu para executar a macro.



76 Parte II: Como o VBA Trabalha com o Excel

As teclas de atalho que você atribuiu às macros sobrescrevem as teclas de atalho internas do Excel. Por exemplo, se atribuir Ctrl + C a uma macro, **CUIDADO!**

o macro, você não pode usar essa tecla de atalho para copiar dados na sua pasta de trabalho. Normalmente, este não é um grande problema, porque o Excel sempre fornece outras maneiras de executar comandos.

Executando um procedimento a partir

de um botão ou forma

Talvez você possa gostar da ideia de executar a macro a partir de um botão (ou qualquer outra forma) em uma planilha. Para designar a macro a um botão, siga esses passos:

1. **Ative uma planilha.**
2. **Adicione um botão a partir do grupo Controles de Forma.**

Para isso escolha Desenvolvedor⇒Controles⇒Inserir (veja a Figura 5-5).

Figura 5-5:

Controles
disponíveis
quando você

clica Inserir

na guia

Desenvolvedor

3. Clique na ferramenta Botão no grupo Controle de Formulário.

4. Arraste na planilha para criar o botão.

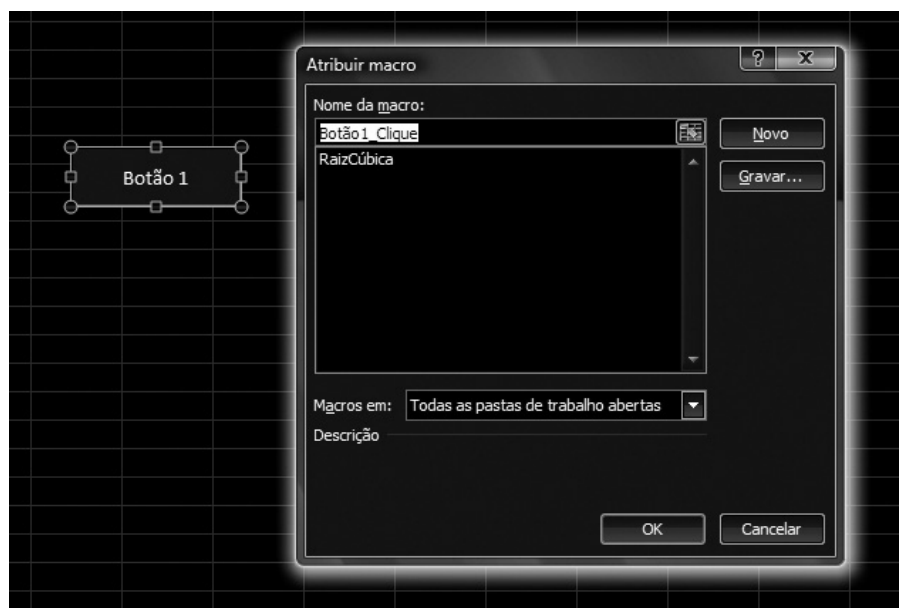
Depois de adicionar o botão à sua planilha, o Excel exibe a caixa de diálogo Atribuir Macro, mostrada na Figura 5-6.

5. Selecione a macro que deseja atribuir ao botão.

6. Clique OK.

ExcelVBABokks.indb 76

15/02/2013 13:59:24



Capítulo 5: Procedimentos Function e Sub no VBA

77

Figura 5-6:

Quando você

acrescentar

um botão a

uma planilha,

automaticamente o Excel

exibe a caixa

de diálogo

Atribuir

Macro.

Depois de ter feito a atribuição, clicar no botão executará a macro — exatamente como uma mágica.

Observe que a caixa de diálogo Atribuir Macro também oferece a oportunidade para gravar uma macro (clicando o botão Gravar). Ou, clique o botão Novo e o Excel inserirá um procedimento Sub vazio, com o nome que você especificar. Mas, na maior parte do tempo, você designará uma macro *existente* a um botão.

Quando você acrescenta um botão, veja que a caixa de opções disponibiliza dois conjuntos de controles: Controles de Formulário e Controles ActiveX. Esses dois grupos de controles são bem parecidos, mas na verdade são bem diferentes. Na prática, os Controles de Formulário são mais fáceis de usar.

Também é possível atribuir uma macro a qualquer outra forma ou objeto. Por exemplo, suponha que você gostaria de executar uma macro quando o usuário clicar em um objeto Retângulo.

1. Adicione o Retângulo à planilha.

Insira um retângulo usando o comando Inserir⇒Ilustrações⇒Formas.

2. Clique no retângulo com o botão direito.

3. Escolha Atribuir Macro em seu menu de atalho.

4. Selecione a macro na caixa de diálogo Atribuir Macro.

5. Clique OK.

Depois de executar essas etapas, ao clicar no retângulo, a macro atribuída será executada.

ExcelVBABokks.indb 77

15/02/2013 13:59:24

78 Parte II: Como o VBA Trabalha com o Excel *Executando um procedimento a partir de*

outro procedimento

Você também pode executar um procedimento a partir de outro procedimento. Se quiser experimentar, acompanhe essas etapas: **1. Ative o módulo VBA que contém a rotina RaizCúbica.**

2. Entre este novo procedimento (acima ou abaixo do código CubeRoot – não faz diferença):

```
Sub NewSub()
```

```
Call RaizCúbica
```

```
End Sub
```

3. Execute a macro NewSub.

A maneira mais fácil de fazer isso é movendo o cursor em qualquer lugar dentro do código NewSub e pressionar F5. Observe que esse procedimento apenas executa o procedimento RaizCúbica.

A propósito, a palavra-chave Call é opcional. A declaração pode consistir apenas do nome do procedimento Sub. Entretanto, eu creio que usar a palavra-chave Call deixa perfeitamente claro que o procedimento está sendo chamado.

Executando Procedimentos Function

Funções, diferentemente de procedimentos Sub, podem ser executadas apenas de duas maneiras:

5 Chamando a função a partir de outro procedimento Sub ou Function.

5 Usando a função em uma fórmula de planilha.

Tente essa simples função. Entre com ela em um módulo VBA:

```
Function RaizCúbica(numero)
```

```
RaizCúbica = numero ^ (1/3)
```

```
End Function
```

Essa função é bem fraca — ela apenas calcula a raiz cúbica do número passado como seu argumento. Entretanto, ela fornece um ponto de partida para entender funções. Ela também ilustra um conceito importante sobre funções: como retornar o valor (você lembra que as funções retornam um valor, certo?).

ExcelVBABokks.indb 78

15/02/2013 13:59:24

Capítulo 5: Procedimentos Function e Sub no VBA

79

Perceba que a única linha do código que cria esse procedimento Function executa um cálculo. O resultado matemático (número para a potência de $1 / 3$) é designado à variável RaizCúbica. Não por coincidência, RaizCúbica também é o nome da função. Para dizer à função qual valor retornar, você designa aquele valor ao nome da função.

Chamando uma função a partir de um

procedimento Sub

Por não ser possível executar essa função diretamente, você deve chamá-la a partir de outro procedimento. Entre com o seguinte procedimento no mesmo módulo VBA que contém a função RaizCúbica.

```
Sub ChamaFunção()
```

```
Ans = RaizCúbica(125)
```

```
MsgBox Ans
```

```
End Sub
```

Quando você executa o procedimento ChamarFunção (usando qualquer dos métodos descritos anteriormente neste capítulo), o Excel exibe uma caixa de mensagem que contém o valor da variável Ans, que é 5.

O que está acontecendo é: a função RaizCúbica é executada e recebe um argumento de 125. O cálculo é realizado pelo código da função e o valor retornado da função é designado à variável Ans. Então, a função MsgBox exibe o valor da variável Ans.

Tente mudar o argumento que é passado para a função RaizCúbica e rode novamente a macro ChamarFunção. Isto funciona como deveria

supondo que você deu à função um argumento válido (um número positivo).

A propósito, o procedimento ChamarFunção pode ser um pouco

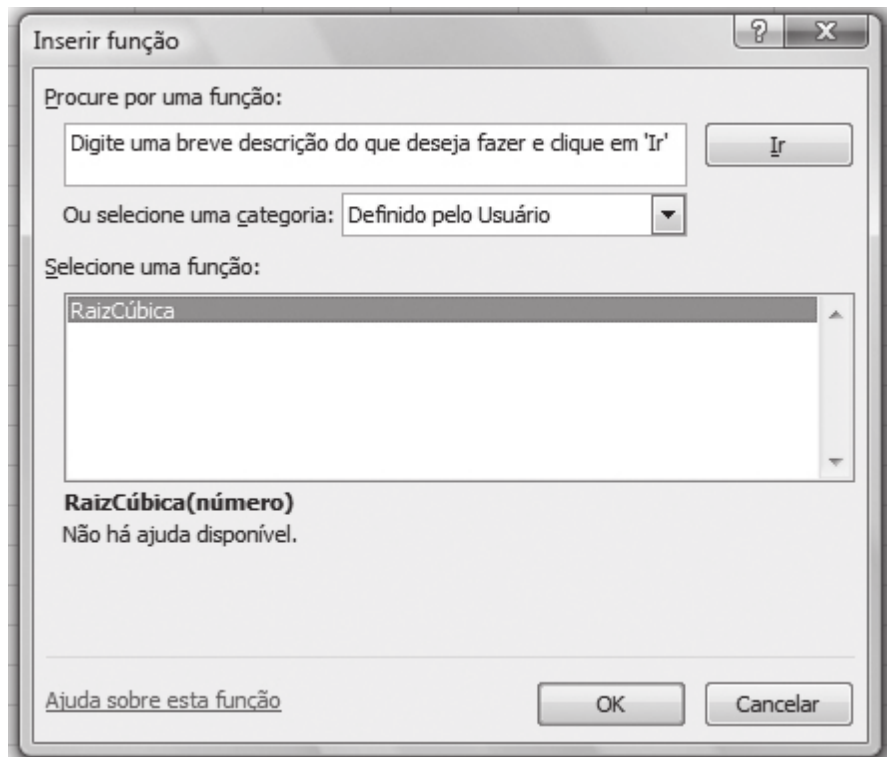
simplificado. A variável Ans não é realmente exigida. Você poderia usar esta única declaração para obter o mesmo resultado:

```
MsgBox RaizCúbica(125)
```

*Chamando uma função a partir de
uma fórmula de planilha*

Agora, é hora de chamar esse procedimento Function do VBA a partir de uma fórmula. Ative uma planilha na mesma pasta de trabalho que contém a definição da função RaizCúbica. Depois entre com a seguinte fórmula em qualquer célula: =RaizCúbica(1728)

A célula Exibirá o número 12, que realmente é a raiz cúbica de 1.728.



80 Parte II: Como o VBA Trabalha com o Excel

Como você deveria esperar, é possível usar uma referência de célula como argumento para a função RaizCúbica. Por exemplo, se a célula A1

contiver um valor, você pode entrar com **=RaizCúbica(A1)**. Nesse caso, a função retorna o número obtido calculando a raiz cúbica do valor em A1.

Você pode usar essa função na planilha quantas vezes quiser. Tal como acontece nas funções internas do Excel, suas funções personalizadas também aparecem na caixa de diálogo Inserir Função. Clique o botão da barra de ferramentas Inserir Função e escolha a categoria User Definido pelo Usuário. Como mostrado na Figura 5-7, a caixa de diálogo Inserir Função relaciona a sua própria função.

Figura 5-7: A

função

RaizCúbica

aparece na

categoria

Definido pelo

Usuário da

caixa de

diálogo

Inserir

Função.

Se você quiser que a caixa de diálogo Insert Function (Inserir Função) **DICA**

exiba uma descrição da função, siga esses passos:

1. Selecione Desenvolvedor⇒Código⇒Macros.

O Excel exibe a caixa de diálogo Macro, mas RaizCúbica não

aparece na lista (RaizCúbica é uma procedimento Function e essa lista mostra apenas procedimentos Sub). Não se aflija.

2. Digite a palavra RaizCúbica na caixa Nome da Macro.

3. Clique no botão Opções.

4. Entre com uma descrição da função na caixa Descrição.

5. Feche a caixa de diálogo Opções de Macro.

6. Feche a caixa de diálogo Macro clicando o botão Cancelar.

Agora, esse texto descritivo aparece na caixa de diálogo

Inserir Função.

Por agora, as coisas devem estar começando a fazer sentido para você (gostaria de ter tido este livro quando *eu* comecei). Você descobriu muito sobre procedimentos Function e Sub. No Capítulo 6, você come-

çará a criar macros. Nele, discute-se os prós e os contras de desenvolver macros usando o gravador de macro do Excel.

15/02/2013 13:59:24

Capítulo 6

Usando o Gravador de

Macro do Excel



Neste Capítulo

- ☐ Gravando suas ações usando o gravador de Macro do Excel
- ☐ Entendendo os tipos de macros que você pode gravar
- ☐ Configurando as opções apropriadas para gravar macro



Você pode usar dois métodos para criar uma macro no Excel: 5
Gravá-la usando o gravador de macro do Excel.

5 Escrevê-la manualmente.

Este capítulo trata especificamente dos prós e contras de usar o gravador de macro do Excel. Gravar uma macro nem sempre é a melhor abordagem, e algumas macros simplesmente não podem ser gravadas, não importa o quanto você tente. Porém, você verá que o gravador de macro do Excel é muito útil. Mesmo que o seu gravador de macro não seja o que você quer, quase sempre ele pode levá-lo na direção certa.

Isto Está Vivo ou É VBA?

Nas edições anteriores deste livro, comparei a gravação de uma macro usando um gravador de fita. No entanto, me ocorreu que os gravadores de fita estão indo rapidamente pelo caminho dos dinossauros. Portanto, modernizei esta seção e agora, a gravação de macro é comparada a fazer um vídeo digital. Essa analogia, como a anterior, não vai muito longe. A Tabela 6-1 compara a gravação de

macro com a feitura de um vídeo.

ExcelVBABokks.indb 81

15/02/2013 13:59:24

82 Parte II: Como o VBA Trabalha com o Excel Tabela 6-1: Gravador de vídeo versus gravador de macro

Gravador de vídeo

Gravador de macro Excel

Qual equipamento

Uma câmera de vídeo.

Um computador e uma

é necessário?

cópia do Excel.

O que é gravado?

Vídeo e áudio.

Ações tomadas em Excel.

Onde a gravação é

Em um cartão de

Em um módulo VBA.

armazenada? memória.

Como você a exibe?

Localiza-se o arquivo e

Localiza-se a macro na caixa

pressiona-se Play.

de diálogo Macros e

clica-se em Executa ou

usa-se

outros métodos.

Você pode editar a

Sim, se tiver o software

Sim, se você souber o que

gravação?

adequado.

está fazendo.

Você pode copiar a

Sim, exatamente como co—

Sim (nenhum equipamento

gravação?

piar qualquer outro arquivo. adicional é exigido).

A gravação é exata?

Depende da situação e da Depende de como você

qualidade do equipamento. configura as coisas ao

gravar a macro.

E se você cometer

Regrave o vídeo (ou, se

Regrave a macro (ou

um erro?

possível, edite-o).

edite-a, se possível).

Você pode ver a

Sim, abrindo o arquivo com Sim, abrindo um módulo no
gravação?

o software adequado.

VBE.

Você pode comparti—

Sim, YouTube é uma boa

Sim, coloque-a em seu site

lhá-la com o mundo?

opção.

ou blog.

Você pode ganhar

Sim, se ela for realmente

Sim, mas você precisa

dinheiro com a

boa (geralmente, é preciso fazer um bocado de edição

gravação? editar).

antes.

O Básico sobre Gravação

Siga os seguintes passos básicos quando gravar uma macro. Mais adiante neste capítulo, eu descrevo esses passos em mais detalhes **1. Determine o que você quer que a macro faça.**

2. Configure adequadamente as coisas.

Essa etapa determina quão bem as suas macros funcionam.

3. Determine se você quer que as referências em sua macro sejam relativas ou absolutas.

ExcelVBABokks.indb 82

15/02/2013 13:59:24

Capítulo 6: Usando o Gravador de Macro do Excel

83

4. Clique no botão Gravar Macro do lado esquerdo da barra de status (ou escolha Desenvolvedor⇒Código⇒Gravar Macro).

O Excel exibe a caixa de diálogo Gravar Macro.

5. Entre com um nome, tecla de atalho, local de macro e descrição.

Cada um desses itens — à exceção do nome — é opcional.

6. Clique OK na caixa de diálogo Gravar Macro.

Automaticamente, o Excel insere um módulo de VBA. Desse ponto em diante, o Excel converte suas ações em código de VBA. Ele

também exibe um botão Parar Gravação em sua barra de status

(um quadrado azul).

7. Execute as ações que você quer gravar usando o mouse ou o teclado.

8. Quando tiver terminado, clique no botão Parar Gravação na barra de status (ou escolha Desenvolvedor⇒Código⇒Parar Gravação).

O Excel para de gravar as suas ações.

9. Teste a macro para garantir que ela funciona corretamente 10. Opcionalmente, você pode limpar o código, removendo declarações estranhas.

O gravador de macro é mais adequado para macros simples, diretas.

Por exemplo, você pode criar uma macro que aplique uma formatação a um intervalo de células ou que configure cabeçalhos de linhas e coluna em uma nova planilha.

BRE-SE

O gravador de macro só é usado para procedimentos Sub. Não é

LEM

possível usá-lo para criar procedimentos Function.

Você também pode considerar o gravador de macro útil para desenvolver macros mais complexas. Frequentemente, gravo algumas ações e depois copio o código gravado em outro, uma macro mais complexa. Na maioria dos casos, é preciso editar o código gravado e adicionar algumas declarações VBA novas.

O gravador de macro *não pode* gerar código para qualquer uma das seguintes tarefas, que descrevo mais adiante no livro:

5 Executar quaisquer tipos de loops de repetição

5 Executar quaisquer tipos de ações condicionais (usando uma declaração If-Then)

5 Atribuir valores a variáveis

5 Especificar tipos de dados

5 Exibir mensagens pop-up

5 Exibir caixas de diálogo personalizadas

ExcelVBABokks.indb 83

15/02/2013 13:59:24

84 Parte II: Como o VBA Trabalha com o Excel A capacidade limitada do gravador de macro certamente não diminui a **BRE-SE**

sua importância. Ao longo do livro, eu afirmo isso: *Gravar as suas ações*, **LEM**

talvez seja a melhor maneira de administrar VBA. Quando estiver em dúvida, tente gravar. Ainda que o resultado possa não ser exatamente o desejado, ver o código gravado pode encaminhá-lo para a direção certa.

Preparação para Gravar

Antes que você dê um grande passo e ligue o gravador de macro, perca um minuto ou dois pensando no que vai fazer. Você grava uma macro para que o Excel possa repetir automaticamente as ações gravadas.

BRE-SE

LEM

Finalmente, o sucesso de uma macro gravada depende de cinco fatores: 5 Como a pasta de trabalho está configurada enquanto você

grava a macro

5 O que está selecionado quando você começa a gravar

5 Se você usa o modo de gravação absoluto ou relativo

5 A exatidão de suas ações gravadas

5 O contexto no qual você exibe a macro gravada

A importância desses fatores ficará clara quando eu o encaminhado através de um exemplo.

Relativo ou Absoluto?

Ao gravar suas ações, normalmente o Excel grava as referências absolutas das células (essa é a modalidade padrão de gravação). Com frequência, esse é o modo *errado* de gravação. Se você usar a gravação relativa, o Excel grava as referências relativas das células. A distinção é explicada nesta seção.

Gravando no modo absoluto

Siga estas etapas para gravar uma macro simples, no modo absoluto.

Essa macro fornece apenas três nomes de mês em uma planilha:

1. Escolha **Desenvolvedor**⇒**Código**⇒**Gravar Macro**.
2. Digite ***Absolute*** como o nome para esta macro.
3. Clique **OK** para começar a gravar.

ExcelVBABokks.indb 84

15/02/2013 13:59:24

Capítulo 6: Usando o Gravador de Macro do Excel

85

4. Ative célula B1 e digite ***Jan*** nesta célula.
5. Mova para a célula C1 e digite ***Fev***.
6. Mova para a célula D1 e digite ***Mar***.
7. Clique a célula B1 para ativar o VBE.

8. Pare o gravador de macro.

9. Pressione Alt + F11 para ativar o VBE.

10. Examine o módulo Module1.

O Excel gera o seguinte código:

```
Sub Absolute()
```

```
“ Absolute Macro
```

```
‘ Range(“B1”).Select
```

```
ActiveCell.FormulaR1C1 = “Jan”
```

```
Range(“C1”).Select
```

```
ActiveCell.FormulaR1C1 = “Fev”
```

```
Range(“D1”).Select
```

```
ActiveCell.FormulaR1C1 = “Mar”
```

```
Range(“B1”).Select
```

```
End Sub
```

Quando executada, esta macro seleciona a célula B1 e introduz os nomes dos três meses nos intervalos B1: D1. Então a macro reativa a célula B1.

Essas mesmas ações ocorrem independente da célula estar ativa quando você executa a macro. Uma macro gravada usando referências absolutas produz sempre os mesmos resultados quando é executada.

Nesse caso, a macro entra sempre os nomes dos primeiros três meses na faixa B1: D1.

Gravando no modo relativo

Em alguns casos, você prefere que a sua macro gravada trabalhe com locais de célula de uma maneira *relativa*. Você pode querer que a macro comece a entrar com os nomes de mês na célula ativa. Em tal caso, é preciso usar a gravação relativa.

Você pode mudar a forma pela qual o Excel grava suas ações, clicando

o botão Usar Referências Relativas no grupo Código, na guia Desenvolvedor. Esse botão é de alternância (liga e desliga). Normalmente, quando o botão aparece, com uma cor diferente, você está gravando no modo relativo. Quando aparece de forma normal, a gravação é absoluta.

BRE-SE

É possível mudar o método de gravação a qualquer momento, mesmo **LEM**

no meio da gravação.

ExcelVBABokks.indb 85

15/02/2013 13:59:25

86 Parte II: Como o VBA Trabalha com o Excel Para ver como funciona o modo relativo de gravação, limpe as células do intervalo B1: D1 e depois execute as seguintes etapas: **1. Ative a célula B1.**

2. Escolha Desenvolvedor⇒Código⇒Gravar Macro.

3. Nomeie essa macro Relativa.

4. Clique Ok para começar a gravar.

5. Clique o botão Usar Referências Relativas para mudar o modo de gravação para relativo.

Ao clicar esse botão, ele muda para uma cor diferente em relação aos outros botões da faixa de opções.

6. Ative a célula B1 e digite Jan nessa célula.

7. Digite Fev na célula C1.

8. Digite Mar na célula D1.

9. Selecione a célula B1.

10. Pare o gravador da macro.

Observe que esse procedimento difere ligeiramente do exemplo anterior. Nesse exemplo, você ativa a célula *antes* de começar a gravar.

Essa é uma etapa importante quando você grava macros que usam a célula ativa como base.

Essa macro sempre inicia entrando com texto na célula ativa. Experimente. Mova o ponteiro para qualquer célula e execute a macro Relativa. Os nomes dos meses são sempre fornecidos no início, na célula ativa.

Com o modo de gravação configurado para relativo, o código que o Excel gera é bem diferente do código gerado no modo absoluto: Sub Relative()

“ Relative Macro

‘ ActiveCell.FormulaR1C1 = “Jan”

ActiveCell.Offset(0, 1).Range(“A1”).Select

ActiveCell.FormulaR1C1 = “Fev”

ActiveCell.Offset(0, 1).Range(“A1”).Select

ActiveCell.FormulaR1C1 = “Mar”

ActiveCell.Offset(0, -2).Range(“A1”).Select

End Sub

Para testar essa macro, ative qualquer célula, exceto B1. Os nomes dos meses são inseridos em três células, começando com a célula que você ativou.

ExcelVBABokks.indb 86

15/02/2013 13:59:25

Capítulo 6: Usando o Gravador de Macro do Excel

87

Observe que o código gerado pelo gravador de macro refere-se à célula A1. Isso pode parecer estranho porque você nunca usou a célula A1

DE ESPECIALISTA

durante a gravação da macro. Este é simplesmente um subproduto de
O P A P

como o gravador de macro funciona (isso é discutido em mais detalhes no Capítulo 8, onde falo sobre o método Offset).

O Que É Gravado?

Quando você liga o gravador de macro, o Excel converte as ações do mouse e do teclado em código válido de VBA. Provavelmente, eu poderia escrever várias páginas descrevendo como o Excel faz isso, mas a melhor maneira de compreender o processo é observando o gravador de macro em ação (a Figura 6-1 mostra como se parecia a minha tela, enquanto o meu gravador de macro estava ligado).

Siga estas etapas:

- 1. Comece com uma pasta de trabalho em branco.**
- 2. Certifique-se de que a janela do Excel não esteja maximizada.**
- 3. Pressione Alt + F11 para ativar o VBE (e para garantir que a janela desse programa não está maximizada).**
- 4. Redimensione e organize a janela do Excel e a janela do VBE de modo que ambas estejam visíveis.**

Para melhores resultados, posicione a janela do Excel acima da janela do VBE e minimize todas os outros aplicativos que estiverem rodando.

- 5. Ative o Excel e escolha Desenvolvedor⇒Código⇒Gravar Macro.**
- 6. Clique Ok para iniciar o gravador de macro.**

O Excel insere um módulo novo (nomeado como Módulo1) e começa a gravar nesse módulo.

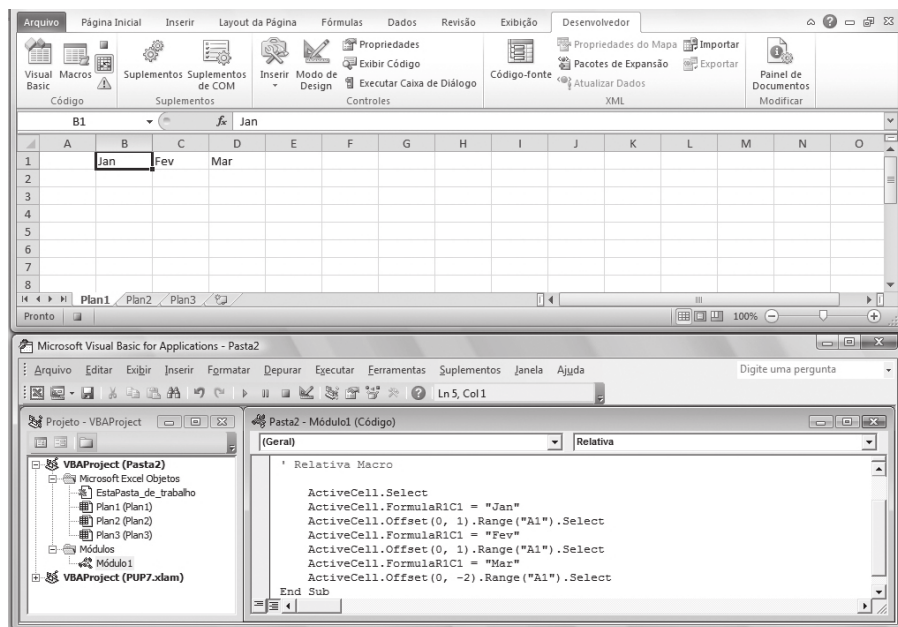
- 7. Ative a janela do programa VBE.**
- 8. Na janela Project Explorer, clique duas vezes Módulo1 para exibir o módulo na janela de Código.**

Retorne ao Excel e brinque um pouco. Escolha vários comandos do Excel e veja o código ser gerado na janela VBE. Selecione células,

entre com dados, formate células, use os comandos da Faixa de Opções, crie um gráfico, altera as larguras de colunas, manipule objetos gráficos e assim por diante – enlouqueça! Eu garanto que você será iluminado enquanto observar o Excel cuspir código VBA diante dos seus olhos.

ExcelVBABokks.indb 87

15/02/2013 13:59:25



88 Parte II: Como o VBA Trabalha com o Excel

Figura 6-1:

Um arranjo
conveniente de
janela,
para
observar o
gravador
de macro

fazer suas

coisas.

Opções da Gravação

Ao gravar suas ações para criar o código VBA, você tem diversas opções. Lembre-se que o comando Desenvolvedor⇒Código⇒Gravar

Macro indica a caixa de diálogo Gravar Macro antes que a gravação comece, como mostrado na Figura 6-2.

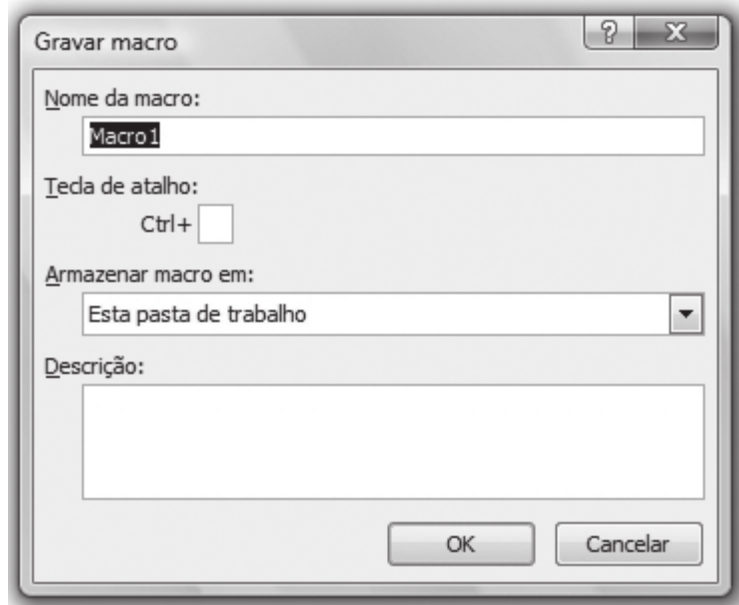
A caixa de diálogo Gravar Macro, mostrada na Figura 6-2, permite que você especifique alguns aspectos de sua macro. Nas próximas seções, descrevo estas opções.

Nome da Macro

Você pode entrar com um nome para o procedimento Sub que você está gravando. Por padrão, o Excel usa os nomes Macro1, Macro2, e assim por diante, para cada macro que você grava. Aceito, geralmente, apenas o nome padrão. Se a macro funciona corretamente e quero salvá-la, mais tarde crio um nome apropriado. Porém, você pode preferir nomear a macro logo no início, a escolha é sua.

ExcelVBABokks.indb 88

15/02/2013 13:59:25



Capítulo 6: Usando o Gravador de Macro do Excel

89

Figura 6-2:

A caixa de

diálogo

Gravar

Macro

oferece

várias

opções.

Tecla de Atalho

A opção Tecla de Atalho permite que você execute a macro pressionando uma combinação de teclas. Por exemplo, se você incorporar **w** (minúscula), pode executar a macro pressionando Ctrl + W. Se entrar com **W** (maiúscula), a macro é ativada quando você

pressionar Ctrl + Shift + W.

Você pode adicionar ou mudar uma tecla de atalho a qualquer momento-BRE-SE

to, assim não é preciso definir essa opção ao gravar uma macro. Veja, no **LEM**

Capítulo 5, instruções sobre a atribuição de uma tecla de atalho a uma macro existente.

Armazenar Macro Em

A opção Armazenar Macro Em informa ao Excel onde armazenar a

macro que ele está gravando. Por padrão, Excel coloca a macro gravada em um módulo na pasta de trabalho ativa. Se preferir, você pode gravá-la em uma nova pasta de trabalho (o Excel abre uma pasta de trabalho em branco) ou em sua Pasta de Trabalho Pessoal de Macros.

A sua Pasta de Trabalho Pessoal de Macros é uma pasta de trabalho oculta que se abre, automaticamente, quando o Excel inicia. Esse é um lugar bom para armazenar as macros que você usará com múltiplas pastas de trabalho. A Pasta de Trabalho Pessoal de Macros é nomeada como PERSONAL.XLSB. Este arquivo não existe até que você o especifique como o lugar para uma macro gravada. Se tiver feito quaisquer alterações nesse arquivo, o Excel solicita que você o salve ao sair.

Descrição

Se você quiser acrescentar alguns comentários descritivos à sua macro, use a caixa Descrição. É possível colocar qualquer coisa que você queira aqui, ou então, nada. Quanto a mim, a opção Descrição é uma perda de tempo, pois é muito fácil acrescentar comentários à macro gravada.

ExcelVBABokks.indb 89

15/02/2013 13:59:25

90 Parte II: Como o VBA Trabalha com o Excel *Essa Coisa É Eficiente?*

Você pode pensar que gravar uma macro renderia alguma premiação

por código VBA – melhor do que você poderia se escrevesse à mão.

Pense de novo. Em muitos casos, o gravador cospe muito lixo e, geralmente, gera código que é menos eficiente.

Não me entenda mal. Sou um leal partidário do gravador de macro.

Trata-se de uma ótima ferramenta para você administrar VBA. Entretanto, diferente de simples macros, eu nunca usei um gravador de macro sem corrigir o código um pouco (geralmente um pouco).

Só para demonstrar o quão ineficiente um código gerado pelo gravador de macro pode ser, tente isso:

1. Ligue o gravador de macro.

2. Escolha o comando Layout da Página⇒Configurar

Página⇒Orientação⇒Paisagem.

3. Desligue o gravador de macro.

Para dar uma olhada na macro, ative a folha Módulo 1. Apenas esse comando — e bem simples — gera o seguinte código:

```
Sub Macro1()
```

```
With ActiveSheet.PageSetup
```

```
.PrintTitleRows = ""
```

```
.PrintTitleColumns = ""
```

```
End With
```

```
ActiveSheet.PageSetup.PrintArea = ""
```

```
With ActiveSheet.PageSetup
```

```
.LeftHeader = ""
```

```
.CenterHeader = ""
```

```
.RightHeader = ""
```

```
.LeftFooter = ""
```

```
.CenterFooter = ""
```

```
.RightFooter = ""  
.LeftMargin = Application.  
InchesToPoints(0.708661417322835)  
.RightMargin = Application.  
InchesToPoints(0.708661417322835)  
.TopMargin = Application.  
InchesToPoints(0.748031496062992)  
.BottomMargin = Application.  
InchesToPoints(0.748031496062992)  
ExcelVBABokks.indb 90  
15/02/2013 13:59:25
```

Capítulo 6: Usando o Gravador de Macro do Excel

91

```
.HeaderMargin = Application.  
InchesToPoints(0.31496062992126)  
.FooterMargin = Application.  
  
InchesToPoints(0.31496062992126)  
.PrintHeadings = False  
.PrintGridlines = False  
.PrintComments = xlPrintNoComments  
.PrintQuality = 300  
.CenterHorizontally = False
```

.CenterVertically = False

.Orientation = xlLandscape

.Draft = False

.PaperSize = xlPaperLetter

.FirstPageNumber = xlAutomatic

.Order = xlDownThenOver

.BlackAndWhite = False

.Zoom = 100

.PrintErrors = xlPrintErrorsDisplayed

.OddAndEvenPagesHeaderFooter = False

.DifferentFirstPageHeaderFooter = False

.ScaleWithDocHeaderFooter = True

.AlignMarginsHeaderFooter = True

.EvenPage.LeftHeader.Text = ""

.EvenPage.CenterHeader.Text = ""

.EvenPage.RightHeader.Text = ""

.EvenPage.LeftFooter.Text = ""

.EvenPage.CenterFooter.Text = ""

.EvenPage.RightFooter.Text = ""

.FirstPage.LeftHeader.Text = ""

.FirstPage.CenterHeader.Text = ""

.FirstPage.RightHeader.Text = ""

.FirstPage.LeftFooter.Text = ""

.FirstPage.CenterFooter.Text = ""

```
.FirstPage.RightFooter.Text = ""
```

```
End With
```

```
End Sub
```

Você deve ter ficado surpreso com a quantidade de código gerado por esse único comando (eu fiquei, na primeira vez em que tentei algo assim). Ainda que você só tenha alterado uma configuração de impressão, o Excel gerou um código que configura muitas outras propriedades relacionadas à impressão.

Esse é um exemplo de gravação de macro exagerada. Se você quiser que uma macro apenas troque a configuração de página para o modo de paisagem, pode simplificar consideravelmente essa macro, apagando o código estranho. Isso torna a macro mais rápida e mais fácil de ler.

Eis como a macro se parece depois que apaguei as linhas irrelevantes:
ExcelVBABokks.indb 91

15/02/2013 13:59:26

92 Parte II: Como o VBA Trabalha com o Excel Sub Macro1()

```
With ActiveSheet.PageSetup
```

```
.Orientation = xlLandscape
```

```
End With
```

```
End Sub
```

Eu apaguei todo o código, exceto a linha que configura a propriedade Orientação. Na verdade, você pode simplificar ainda mais essa macro, porque de fato você não precisa da declaração With.End (eu explico mais sobre isso no Capítulo 14): Sub Macro1()

```
ActiveSheet.PageSetup.Orientation = xlLandscape
```

```
End Sub
```

Nesse caso, a macro muda a propriedade Orientação do objeto Configuração de Página na folha ativa. Todas as outras propriedades permanecem inalteradas. A propósito, xlLandscape é uma constante

interna que torna o seu código mais fácil de ler. Essa constante tem um valor de 2, portanto, a seguinte declaração funciona exatamente da mesma forma (mas, não é tão fácil de ler):

```
ActiveSheet.PageSetup.Orientation = 2
```

Fique atento. Eu discuto constantes internas no Capítulo 7.

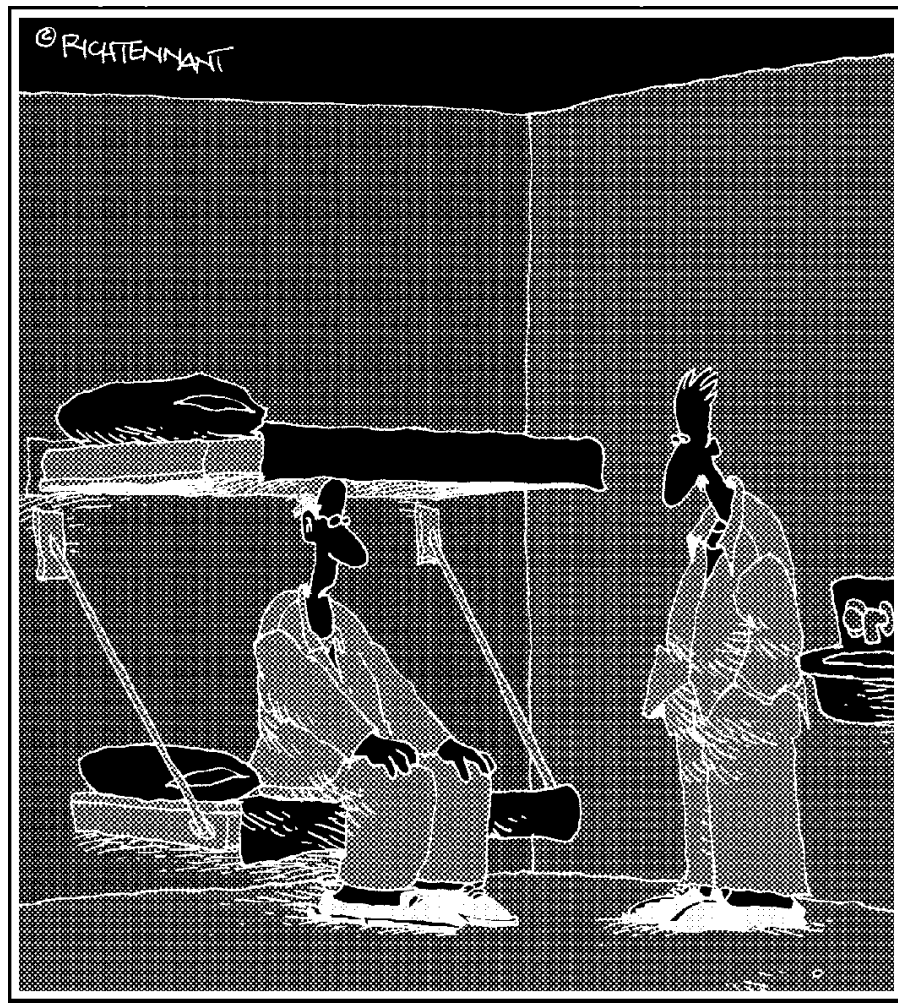
Ao invés de gravar essa macro, você pode entrar diretamente com ela em um módulo VBA. Para isso, você tem que saber quais objetos, propriedades e métodos usar. Embora a macro gravada não seja tão boa, gravando-

-a você percebe que o objeto Configuração de Página tem uma propriedade Orientação. Munido com esse conhecimento (e provavelmente, algumas tentativas) você pode escrever manualmente a macro.

Este capítulo é quase um resumo de como usar o gravador de macro. A única coisa que falta é experiência. Ocasionalmente, você descobre quais declarações gravadas você pode apagar com segurança. Melhor ainda, você descobre como modificar uma macro gravada para torná-la mais útil.

ExcelVBABokks.indb 92

15/02/2013 13:59:26



Parte III

Conceitos de

Programação

A 5ª Onda

Por Rich Tennant

“Eu comecei pensando em cenas do tipo ‘e se’
na minha planilha, ‘E se eu ficasse enjoado,
desse meu trabalho com dinheiro sujo e

encaminhasse um pouco do dinheiro da
empresa para uma conta em
um paraíso fiscal?””

ExcelVBABokks.indb 93

15/02/2013 13:59:26

Nesta parte...

E sta é a parte do livro que você tem esperado.

Nos próximos oito capítulos, você descobrirá
tudo sobre os elementos essenciais de programação
do Excel. E, no decorrer do processo, verá alguns
exemplos esclarecedores que pode adaptar às suas
próprias necessidades.

ExcelVBABokks.indb 94

15/02/2013 13:59:26

Capítulo 7

Elementos Essenciais da Linguagem VBA



Neste Capítulo

- ☐ Saber quando, por que e como usar comentários em seu código
- ☐ Usar variáveis e constantes
- ☐ Informar ao VBA qual tipo de dados você está usando
- ☐ Saber porque você precisa usar rótulos em seus procedimentos





Pelo fato do VBA ser uma linguagem de programação real e viva, ele usa muitos elementos comuns a todas as linguagens de programação. Neste capítulo, eu o apresento a vários desses elementos: comentários, variáveis, constantes, tipos de dados, arrays e algumas outras benesses. Se você tiver programado com outras linguagens, um pouco desse material será familiar. Se você for um programador iniciante, é hora de arregalar as mangas e ficar ocupado.

Usando Comentários em Seu Código VBA

Um *comentário* é o tipo mais simples de declaração VBA. Porque o VBA ignora essas declarações, elas podem não conter nada do que você quer. É possível inserir um comentário para se lembrar do motivo pelo qual fez algo ou para esclarecer algum código especialmente elegante que escreveu. Use comentários generosa e extensamente para descrever o que o código faz (o que ele não faz é sempre óbvio pela leitura do próprio código). Frequentemente, um código que faz todo o sentido nos dias de hoje o mistifica amanhã. Esteja lá. Faça isso.

Um comentário é iniciado com um apóstrofo ('). O VBA ignora qualquer texto que siga um apóstrofo em uma linha de código. Você pode usar uma linha inteira para o seu comentário ou inserir o seu comentário ao final de uma linha de código. O exemplo a seguir mostra um procedimento VBA com três comentários, ainda que eles não sejam, necessariamente, *bons* comentários.

ExcelVBABokks.indb 95

15/02/2013 13:59:26

96 Parte III: Conceitos de Programação

```
'Sub CommentsDemo()
```

```
Esse procedimento não faz nada de valor
```

```
x = 0 'x não representa nada
```

```
'mostra o resultado
```

```
MsgBox x
```

```
End Sub
```

A regra “o apóstrofo indica um comentário” tem uma exceção. VBA não **BRE-SE**

interpreta um apóstrofo entre aspas como uma indicação de comentá-

LEM

rio. Por exemplo, a seguinte declaração não contém um comentário, ainda que ela tenha um apóstrofo:

Msg = “Can’t continue”

Quando estiver escrevendo código, você pode querer testar um procedimento *excluindo* uma declaração em especial, ou grupo de declarações.

Você *poderia* apagar as declarações e, mais tarde, digitá-las novamente.

Mas isso é uma perda de tempo. Uma solução melhor é apenas transformar aquelas declarações em comentários, inserindo apóstrofos. VBA ignora declarações iniciadas com apóstrofos ao executar uma rotina. Para reativar aquelas declarações “comentadas”, basta remover os apóstrofos.

Eis uma maneira rápida de converter um bloco de declarações a comen-DICA

tários. No VBE, escolha Exibir⇒Barras de ferramentas⇒Editar para exibir a barra de ferramentas Editar. Para converter um bloco de declarações a comentários, selecione a declaração e clique o botão Comentar Bloco. Para remover os apóstrofos, selecione as declarações e clique o botão Remover Comentário do Bloco.

Embora comentários possam ser úteis, nem todos os comentários são criados da mesma forma. Por exemplo, o seguinte procedimento usa muitos comentários, porém, eles não acrescentam nada útil. Neste caso, o código é claro o bastante sem os comentários.

Sub BadComments()

‘ Declara variáveis

Dim x As Integer

Dim y As Integer

Dim z As Integer

‘ Inicia a rotina

x = 100 ‘ Assign 100 to x

y = 200 ‘ Assign 200 to y

‘ Adiciona x e y e os coloca em z

z = x + y

‘ Mostra o resultado

MsgBox z

End Sub

Cada pessoa desenvolve o seu próprio estilo de comentar. No entanto, para ser útil, os comentários devem apresentar as informações que não são óbvias a partir da leitura do código. Caso contrário, os comentários só mastigam bytes e tornam os arquivos maiores do que o necessário.

ExcelVBABokks.indb 96

15/02/2013 13:59:26

Capítulo 7: Elementos Essenciais da Linguagem VBA

97

As seguintes dicas podem ajudá-lo a usar comentários com eficiência:

DICA

5 Descreva resumidamente cada procedimento Sub ou Function que escrever.

5 Use comentários para controlar as mudanças que você fez em um procedimento.

5 Use um comentário para indicar que você está usando uma função ou uma declaração de maneira incomum ou fora do padrão.

5 Use comentários para descrever as variáveis que usar, especialmente se você não usar nomes significativos para as variáveis.

5 Use um comentário para descrever quaisquer soluções que você desenvolva para superar bugs no Excel.

5 Escreva comentários enquanto desenvolve código ao invés de guardar a tarefa para uma etapa final.

5 Dependendo do seu ambiente de trabalho, considere adicionar uma ou duas piadas como um comentário. As pessoas que assumir em seu trabalho quando você for promovido podem apreciar o humor.

Usando Variáveis, Constantes e

Tipos de Dados

O principal objetivo do VBA é manipular dados. O VBA armazena os dados na memória do seu computador: eles podem ou não terminar em um disco. Alguns dados, tais como listagens em planilhas, ficam em objetos. Outros dados são armazenados nas variáveis que você cria.

Entendendo variáveis

Uma variável é apenas um local de armazenagem, nomeado, na memó-

ria do seu computador. Você tem muita flexibilidade na nomeação de

suas variáveis, portanto, torne os nomes de variáveis tão descritivos quanto possível. Você atribui um valor a uma variável usando o sinal de operador de igual (mais sobre isso, mais adiante, na seção “Usando Declarações de Atribuição”).

Os nomes de variáveis nesses exemplos aparecem em ambos os lados, esquerdo e direito, dos sinais de igual. Observe que o último exemplo usa duas variáveis:

```
x = 1
```

```
InterestRate = 0.075
```

```
LoanPayoffAmount = 243089
```

```
DataEntered = False
```

```
x = x + 1
```

```
UserName = “Bob Johnson”
```

```
DataStarted = #3/14/2010#
```

```
MyNum = YourNum * 1.25
```

```
ExcelVBABokks.indb 97
```

```
15/02/2013 13:59:26
```

98 Parte III: Conceitos de Programação O VBA impõe algumas regras com relação aos nomes de variáveis: 5 Você pode usar letras, números e alguns caracteres de pontuação, mas o primeiro caractere deve ser uma letra.

5 Não é possível usar quaisquer espaços ou pontos no nome de uma variável.

5 O VBA não distingue entre letras maiúsculas e minúsculas.

5 Você não pode usar os seguintes caracteres no nome de uma

variável: #, \$, %, & ou !.

5 Os nomes de variáveis não podem ter mais de 255 caracteres. É

claro que você está procurando problemas se usar nomes de variáveis com 255 caracteres de extensão.

Para tornar os nomes de variáveis mais legíveis, frequentemente os programadores misturam letras maiúsculas com minúsculas (por exemplo, InterestRate) ou usam o caractere underline (interest_rate).

O VBA tem muitas palavras reservadas que você não pode usar em nomes de variáveis ou nomes de procedimento. Essas incluem palavras tais como Sub, Dim, With, End, Next e For. Se você tentar usar uma dessas palavras como uma variável, pode receber um erro de compilação (significando que o seu código não vai rodar). Portanto, se ao declarar uma atribuição você obtiver uma mensagem de erro, confira se o nome da variável não é uma palavra reservada.

O VBA permite que você crie variáveis cujos nomes correspondem aos nomes usados no modelo de objeto do Excel, tais como Workbook e Range. Mas, claro, usar nomes como esses só aumenta a possibilidade de tornar as coisas confusas. Assim, resista à tentação de nomear a sua variável como Workbook e, ao invés, use algo como MyWorkbook.

O que são tipos de dados do VBA?

Quando falo sobre *tipos de dados*, estou me referindo à forma pela qual um programa armazena dados na memória – por exemplo, como

números inteiros, números reais ou strings. Ainda que o VBA possa cuidar automaticamente desses detalhes, ele impõe um custo (não existe almoço grátis). Permitir que o VBA cuide dos dados que você vai digitar resulta em execução mais lenta e uso ineficiente de memória. Em pequenos aplicativos, normalmente isso não representa um grande problema. Porém, em aplicativos grandes ou complexos, os quais podem ficar lentos ou precisar de proteção a cada byte de memória, você precisa estar familiarizado com os tipos de dados.

Automaticamente, VBA lida com todos os detalhes quanto aos dados, o que facilita a vida para os programadores. Nem todas as linguagens de programação oferecem esse luxo. Por exemplo, algumas linguagens

são *digitadas estritamente*, significando que o programador deve definir explicitamente o tipo de dados em cada variável usada.

VBA não exige que você declare as variáveis que você usa, mas, definitivamente, essa é uma boa prática. Mais adiante, neste capítulo, você verá o motivo.

ExcelVBABokks.indb 98

15/02/2013 13:59:27

Capítulo 7: Elementos Essenciais da Linguagem VBA

99

O VBA tem uma variedade de tipos de dados integrados. A Tabela 7-1 relaciona os tipos mais comuns de dados com os quais o VBA pode lidar.

Tabela 7-1

Tipos de Dados Integrados do VBA

Tipo de Dados

Bytes Usados Faixa de Valores

Boolean

2

Verdadeiro ou Falso

Integer

2

-32.768 a 32.767

Long

4

-2.147.483.648 a 2.147.483.647

Single

4

-3.402823E38 a 1.401298E45

Double (negativo)

8

-1.79769313486232E308 a -4.94065645841247E-324

Double (positivo)

8

4.94065645841247E-324 a 1.79769313486232E308

Currency (Moeda)

8

-922.337.203.685.477.5808 a 922.337.203.685.477.5807

Date

8

1/1/100 a 12/31/9999

String

1 por caractere Varia

Object

4

Qualquer objeto definido

Variant

Varia

Qualquer tipo de dados

User Defined

Varia

Varia

Em geral, escolha o tipo de dados que usa a menor quantidade de bytes, mas que ainda pode lidar com todos os dados que você deseja armazenar na variável.

Geralmente, os contadores de loop são declarados como inteiros. Se você usar um contador de loop para contar as linhas de uma planilha, o **CUIDADO!**

seu programa pode simplesmente acusar erro! Por quê? Integers não podem ser maiores que 32.767. A partir do Excel 2007, as planilhas têm muito mais linhas (1.048.576, para ser exato). Em lugar disso, declare esses contadores de loop como Long.

Declarando e estendendo variáveis

Se você leu as seções anteriores, agora sabe um pouco sobre variáveis e tipos de dados. Nesta seção, você descobre como declarar uma variável como determinado tipo de dados.

Se você não declarar o tipo de dados em uma variável que usar em uma rotina VBA, o VBA usa o tipo de dados padrão: Variant. Dados armazenados como uma variante agem como um camaleão: eles mudam dependendo do que você fizer com eles. Por exemplo, se uma variável for um tipo de dados Variant e contiver uma string de texto que se ExcelVBABokks.indb 99

15/02/2013 13:59:27

100 Parte III: Conceitos de Programação parece com um número (tal como “143”), você pode usar essa variável tanto como uma string quanto como um número calculável. Automaticamente, o VBA lida com a conversão. Deixar que o VBA lide com os tipos de dados pode parecer uma saída fácil — mas lembre-se de que você sacrifica a velocidade e a memória.

Antes de usar variáveis em um procedimento, é uma excelente ideia *declarar* as suas variáveis – isto é, informar ao VBA o tipo de dados de cada variável. Declarar as suas variáveis faz o seu programa rodar mais depressa e usar a memória com mais eficiência. O tipo de dados padrão, Variant, leva o VBA a executar repetidamente verificações demoradas e reserva mais memória do que necessário. Se o VBA conhecer o tipo de dados de uma variável, ele não precisa investigar

nada e pode reservar memória suficiente apenas para armazenar os dados.

Para impor a si mesmo a tarefa de declarar todas as variáveis que usa, inclua o seguinte como a primeira declaração em seu módulo VBA:
Option Explicit

É preciso usar Option Explicit apenas uma vez: no início de seu módulo, **BRE-SE**

antes da declaração de quaisquer procedimentos no módulo. Tenha em **LEM**

mente que a declaração Option Explicit só se aplica ao módulo no qual ela reside. Se você tiver mais que um módulo VBA em um projeto, precisa incluir essa declaração em cada módulo.

Imagine que você usa uma variável não declarada (isto é, uma Variant) chamada CurrentRate. Em algum lugar em sua rotina, você insere a seguinte declaração:

```
CurentRate = .075
```

Esta variável possui um erro ortográfico, e provavelmente levará a sua rotina a resultados incorretos. Se você usar Option Explicit no início do seu módulo (forçando você a declarar a variável Current Rate), o Excel gera um erro se ele encontrar algum erro de grafia naquela variável.

Para garantir que a declaração Option Explicit entre automaticamente **DICA**

quando você inserir um novo módulo VBA, ative a opção Requerer Definição de Variável. Você a encontrará na guia Editor da caixa de diálogo Opções (no VBE, escolha Ferramentas\Opções. Enfaticamente eu recomendo fazer isso.

Declarar as suas variáveis também permite que você usufrua vantagens de **DICA**

um atalho que pode poupar alguma digitação. Digite apenas os dois ou três primeiros caracteres do nome da variável e depois pressione Ctrl + Space. O

VBE ou completará a entrada para você ou — se a escolha for ambígua

— exibirá uma lista de palavras para seleção. Na verdade, essa dica também funciona com palavras reservadas e com funções.

ExcelVBABokks.indb 100

15/02/2013 13:59:27

Capítulo 7: Elementos Essenciais da Linguagem VBA 101

Agora você conhece as vantagens de declarar variáveis, mas *como* fazer isso? A maneira mais comum é usar uma declaração Dim. Eis alguns exemplos de variáveis sendo declaradas: Dim YourName As String

Dim AmountDue As Double

Dim RowNumber As Long

Dim X

As primeiras três variáveis são declaradas como tipo específico de dados. A última, X, não é declarada como um tipo específico de dados, portanto, ela é tratada como uma Variant (ela pode ser qualquer coisa).

Além de Dim, o VBA tem três outras palavras chave que são usadas para declarar variáveis:

5 Static

5 Public

5 Private

Eu explico mais sobre as palavras chave Dim, Static, Public e Private mais adiante. Primeiro, preciso abordar dois tópicos que são relevantes agora: o escopo de uma variável e a vida de uma variável.

Lembre-se de que uma pasta de trabalho pode ter qualquer quantidade de módulos VBA. E um módulo VBA pode ter qualquer quantidade de procedimentos Sub e Function. O *escopo* de uma variável determina quais módulos e procedimentos podem usar a variável. A Tabela 7-2

descreve os escopos em detalhes.

Confuso? Continue virando as páginas e você verá alguns exemplos que tornarão essa coisa clara como um cristal.

Tabela 7-2

Escopo da Variável

Escopo

Como a Variável é Declarada

Apenas procedimento

Usando uma declaração Dim ou Static no proce-

dimento que usar a variável

Apenas módulo

Usando uma declaração Dim ou Private antes da pri-

meira declaração Sub ou Function no módulo

Todos os procedimentos

Usando uma declaração Public antes da primeira

em todos os módulos

declaração Sub ou Function em um módulo

ExcelVBABokks.indb 101

15/02/2013 13:59:27

102 Parte III: Conceitos de Programação Declarações humorísticas

Tópicos como variáveis, tipos de dados, declarações e escopo podem ser bem monótonos.

Assim, eu reuni algumas declarações meio humorísticas para a sua diversão. Estas são todas declarações válidas:

Dim King As String, Kong As Long

Dim Mouthful as Byte

Dim Julian As Boolean

Dim Unmarried As Single

Dim Trouble As Double

Dim WindingRoad As Long

Dim Blind As Date

Public

Nuisance

Private

FirstClass

Static Cling, Electricity

Dim BaseballCards As New Collection

Dim DentalFloss As String

Provavelmente, você pode encontrar algumas outras.

Variáveis apenas de procedimento

O nível mais baixo de escopo de uma variável está no nível de

procedimento (um *procedimento* Sub ou Function). Variáveis declaradas com esse escopo só podem ser usadas no procedimento em que elas são declaradas. Quando o procedimento terminar, a variável não existe mais (ela vai para o grande buquê no céu), e o Excel libera a sua memória. Se você executar novamente o procedimento, a variável revive, mas o seu valor anterior é perdido.

Uma maneira mais comum de declarar uma variável apenas de procedimento é com uma declaração Dim. Dim não se refere à capacidade mental dos designers de VBA. Ao contrário, trata-se de um antigo termo de programação que é o diminutivo de *dimensão*, significando apenas que você está alocando memória para uma variável em especial.

Geralmente, você coloca declarações Dim logo depois da declaração Sub ou Function e antes do código do procedimento.

O seguinte exemplo mostra algumas variáveis apenas de procedimento, declaradas usando declarações Dim:

```
Sub MySub()
```

```
Dim x As Integer
```

```
Dim First As Long
```

```
Dim InterestRate As Single
```

```
Dim TodaysDate As Date
```

```
Dim UserName As String
```

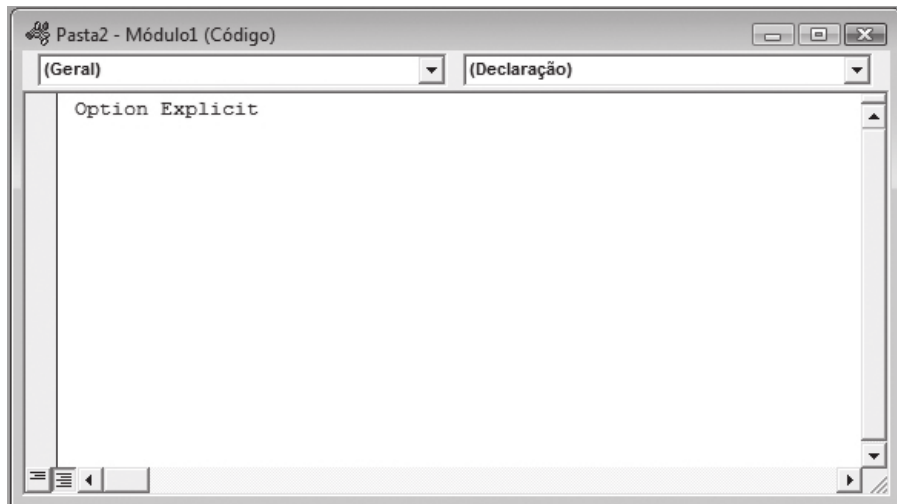
```
Dim MyValue
```

```
‘ ... [O código entra aqui] ...
```

```
End Sub
```

ExcelVBABokks.indb 102

15/02/2013 13:59:28



Capítulo 7: Elementos Essenciais da Linguagem VBA 103

Observe que a última declaração `Dim` no exemplo anterior não declara o tipo de dados; ela declara apenas a própria variável. O efeito é que a variável `MyValue` é uma `Variant`.

A propósito, também é possível declarar diversas variáveis com uma única declaração `Dim`, como no exemplo a seguir:

```
Dim x As Integer, y As Integer, z As Integer
```

```
Dim First As Long, Last As Double
```

Diferente de algumas linguagens, VBA não permite que você declare um grupo de variáveis para ser um tipo especial de dados, separando as **CUIDADO!**

variáveis com vírgulas. Por exemplo, ainda que válida, a seguinte declaração *não* declara todas as variáveis como `Integers` (inteiros):
`Dim i, j, k As Integer`

Neste exemplo, apenas `k` é declarado como um `Inteiro`; as outras variáveis são declaradas como `Variant`.

Se você declara uma variável com escopo apenas de procedimento, outros procedimentos no mesmo módulo podem usar o mesmo nome da variável, porém, cada cópia da variável é única para o seu próprio procedimento. Normalmente, as variáveis declaradas ao nível de

procedimento são as mais eficientes, pois VBA libera a memória que elas usam quando o procedimento termina.

Variáveis apenas para módulo

Às vezes, você quer que uma variável esteja disponível a todos os procedimentos em um módulo. Sendo assim, declare apenas a variável (usando Dim ou Private) *antes* da primeira declaração Sub ou Função do módulo – fora de quaisquer procedimentos. Isso é feito na seção Declaração, no início de seu módulo (é também onde a declaração Option Explicit está localizada). A Figura 7-1 mostra como você sabe quando está trabalhando com a seção Declaração.

Figura 7-1:

Cada módulo

VBA tem

uma seção

Declara-

ções, que

aparece

antes de

quaisquer

procedimentos Sub ou

Function.

ExcelVBABokks.indb 103

15/02/2013 13:59:28

104 Parte III: Conceitos de Programação Como um exemplo, suponha que você queira declarar a variável

CurrentValue para que ela esteja disponível a todos os procedimentos em seu módulo. Tudo o que é preciso fazer é usar a declaração Dim na seção Declarações:

Dim CurrentValue As Integer

Com essa declaração no lugar apropriado, a variável CurrentValue pode ser usada a partir de qualquer outro procedimento dentro do módulo e ela retém o seu valor de um procedimento para outro.

Variáveis Públicas

Se você precisa tornar essa variável disponível para todos os procedimentos em todos os módulos VBA de uma pasta de trabalho, declare essa variável no nível do módulo (na seção Declaração) usando a palavra-chave Public. Aqui está um exemplo: Public CurrentRate As Long

A palavra-chave Public disponibiliza a variável CurrentRate a qualquer procedimento na pasta de trabalho – mesmo aqueles presentes em outros módulos VBA. É preciso inserir essa declaração antes da primeira declaração Sub ou Function em um módulo.

Se você quiser que uma variável esteja disponível para módulos em outras pastas de trabalho, é preciso declarar a variável como pública e

DE ESPECIALISTA

estabelecer uma referência à pasta de trabalho que contém a declara-
O P A P

ção de variável. Configure uma referência usando o comando do VBE

Ferramentas⇒Referências. Na prática, dificilmente é feito o compartilhamento de uma variável por pastas de trabalho. Na verdade, eu nunca fiz isso em toda a minha carreira de programação em VBA. Mas imagino que seja bom saber como isso pode ser feito, no caso de surgir uma situação de risco.

Variáveis Estáticas

Geralmente, quando um procedimento termina, todas as variáveis do procedimento são restabelecidas. *Static variables* são um caso especial, pois elas retêm seus valores mesmo quando o procedimento termina. Você declara uma variável estática ao nível de procedimento. Uma variável estática pode ser útil se você precisar rastrear a quantidade de vezes que executa um procedimento. É possível declarar uma variável estática e aumentá-la cada vez que executar o procedimento.

Como mostrado no exemplo a seguir, variáveis estáticas são declaradas usando a palavra chave Static:

```
Sub MySub()
```

```
Static Counter As Integer
```

```
Dim Msg As String
```

```
Counter = Counter + 1
```

```
Msg = “Números de execuções: “ & Counter
```

```
MsgBox Msg
```

```
End Sub
```

ExcelVBABokks.indb 104

15/02/2013 13:59:28

Capítulo 7: Elementos Essenciais da Linguagem VBA 105

O código controla a quantidade de vezes em que o procedimento foi executado. O valor da variável Counter (contador) não é redefinido quando o procedimento termina, mas quando você fecha e reabre a pasta de trabalho.

Ainda que o valor de uma variável declarada como estática seja retido depois da variável terminar, aquela variável fica indisponível a outros **CUIDADO!**

procedimentos. No exemplo anterior de procedimento, MySub, a variável Counter e o seu valor só estão disponíveis dentro do procedimento MySub. Em outras palavras, ela é uma variável ao nível de procedimento.

Duração das variáveis

Nada dura para sempre, inclusive variáveis. O escopo de uma variável não apenas determina onde aquela variável pode ser usada, mas também afeta em quais circunstâncias a variável é removida da memória.

Você pode remover todas as variáveis da memória usando três métodos: 5 Clique o botão Redefinir na barra de ferramentas (o pequeno botão quadrado azul na barra de ferramentas padrão do VBE).

5 Clique Terminar quando aparecer uma caixa de diálogo com uma mensagem de erro em tempo de execução.

5 Inclua uma declaração End em qualquer lugar de seu código. Isso não é o mesmo que uma declaração End Sub ou End Function.

Caso contrário, apenas variáveis ao nível de procedimento serão removidas da memória quando o código de macro tiver concluído a execução.

Variáveis estáticas, variáveis ao nível de módulo e variáveis globais (públicas), todas retêm seus valores durante a execução de seu código.

Se você usa variáveis ao nível de módulo ou ao nível global, assegure-se de que elas tenham o valor que você espera que tenham. Nunca se sabe **CUIDADO!**

se uma das situações que acabei de mencionar pode ter levado as suas variáveis a perder seu conteúdo!

Trabalhando com constantes

O valor de uma variável pode mudar (e normalmente muda) enquanto o seu procedimento está em execução. Por isso é que ela é chamada de *variável*. Às vezes, você precisa fazer referência a um valor ou string que nunca muda. Nesse caso, você precisa de uma *constante* – um elemento nomeado cujo valor não muda.

Conforme mostrado nos seguintes exemplos, constantes são declaradas usando a declaração Const. A declaração afirmativa também dá à constante o seu valor.

```
Const NumQuarters As Integer = 4
```

```
Const Rate = .0725, Period = 12
```

```
Const ModName As String = "Budget Macros"
```

```
Public Const AppName As String = "Budget Application"
```

106 Parte III: Conceitos de Programação Usar constantes no lugar de valores ou strings bem codificados é uma **DICA**

ótima prática de programação. Por exemplo, se o seu procedimento precisar fazer referência, repetidamente, a um valor específico (tal como uma taxa de juros), é melhor declarar o valor como uma constante e fazer referência ao seu nome, ao invés do valor. Isso torna o seu **Figura 7-2: código** mais legível e mais fácil de alterar. Quando a taxa de juros Constantes

mudar, você só precisará mudar uma declaração, ao invés de várias.

BRE-SE

que

LEM

Como as variáveis, as constantes têm um escopo. Tenha o seguinte em mente: determinam

o modo

5 Para disponibilizar uma constante apenas dentro de um único procedimento, declare a constante depois da declaração do

Calculation.

procedimento Sub ou Function.

5 Para tornar a constante disponível a todos os procedimentos em um módulo, declare a constante na seção Declaração para o

módulo.

5 Para tornar uma constante disponível a todos os módulos na pasta de trabalho, use a palavra chave Public e declare a

constante na seção Declaração de qualquer módulo.

Se você tentar alterar o valor de uma constante em uma rotina VBA, obterá um erro. Isso não é muito surpreendente, pois uma constante é uma constante. Diferente de uma variável, o valor de uma constante não varia. Se precisar alterar o valor de uma constante enquanto o seu código estiver rodando, o que você precisa, realmente, é de uma variável.

Constantes pré-fabricadas

O Excel e o VBA contêm muitas constantes pré-definidas, as quais podem ser usadas sem que você precise declará-las. Normalmente, o gravador de macro usa constantes ao invés de valores atuais. Em geral, você não precisa saber o valor dessas constantes para usá-las. O simples procedimento a seguir usa a constante integrada `xlCalculationManual`, para mudar a propriedade `Calculation` do objeto `Application` (em outras palavras, isso muda o modo de recalculiar do Excel para manual).

```
Sub CalcManual()
```

```
Application.Calculation = xlCalculationManual
```

```
End Sub
```

Eu descobri a constante `xlCalculationManual` gravando uma macro enquanto eu alterava o modo de cálculo. Eu também poderia ter olhado no sistema de Ajuda. A Figura 7-2 exibe a tela de Ajuda que relaciona as constantes para a propriedade Cálculo.

O valor atual da constante `xlCalculationManual` integrada é -4135. Obviamente, é mais fácil usar o nome da constante do que tentar lembrar um número tão estranho. A propósito, a constante para mudar para o modo automático de cálculo é `xlCalculationAutomatic`; o seu valor é -4105. Como é possível ver, muitas das constantes integradas são apenas números arbitrários que têm significado especial no VBA.

ExcelVBABokks.indb 106

15/02/2013 13:59:29



Capítulo 7: Elementos Essenciais da Linguagem VBA 107

Para encontrar o valor atual de uma constante integrada, use a janela **DICA**

Verificação Imediata no VBE e execute uma declaração VBA, tal como a seguinte:

```
? xlCalculationAutomatic
```

Se a janela de Verificação Imediata não estiver visível, clique Ctrl + G. O

ponto de interrogação é uma forma de abreviar a instrução **Print**.

Trabalhando com strings

O Excel pode trabalhar tanto com números quanto com texto, portanto, não é de surpreender que o VBA tenha esse mesmo poder. Geralmente, um texto é referenciado como uma *string* (sequência de caracteres).

Você pode trabalhar em VBA com dois tipos de strings:

5 Strings de extensão fixa são declaradas com uma quantidade específica de caracteres. A extensão máxima é de 65.526 caracteres.

Isso é um monte de caracteres! Como comparação, este capítulo

contém cerca da metade desses caracteres.

5 Strings de extensão variável teoricamente podem conter dois bilhões de caracteres. Se você digitar cinco caracteres por

segundo, levaria 760 dias para atingir dois bilhões de caracteres

– supondo que você não faça pausa para comer ou dormir.

Ao declarar uma string variável com uma declaração Dim, você pode especificar a extensão máxima, se conhecê-la (ela é uma string de extensão fixa) ou deixar que o VBA cuide dela dinamicamente (uma string de extensão variável). O exemplo a seguir declara a variável MyString como uma string com um comprimento máximo de 50 caracteres (use um asterisco para especificar a quantidade de caracteres, até o limite de 65.526 caracteres). YourString também é declarada como uma string, mas o seu comprimento não é especificado: Dim MyString As String * 50

Dim YourString As String

ExcelVBABokks.indb 107

15/02/2013 13:59:29

108 Parte III: Conceitos de Programação Ao declarar uma string de extensão fixa, não use uma vírgula no número que especifica o tamanho da string. Na verdade, nunca use vírgulas ao **CUIDADO!**

entrar com um número de valor em VBA. O VBA não gosta disso.

Trabalhando com datas

Um outro tipo de dados que você pode julgar útil é Date. É possível usar uma string variável para armazenar datas, mas, você não pode realizar cálculos com datas. Usar esse tipo de dados lhe oferece rotinas flexíveis. Por exemplo, você pode calcular o número de dias entre duas datas, o que seria impossível se você usasse strings para conter as suas datas.

Uma variável definida como Date pode conter datas variando de 1º de Janeiro de 0100 a 31 de Dezembro de 9999. Isso é uma expansão de quase 10.000 anos e mais do que o suficiente até mesmo para a previsão financeira mais agressiva. Você também pode usar o tipo de

dados Date para trabalhar com dados de horário (vendo como falta um tipo de dados de horário em VBA).

Estes exemplos declaram variáveis e constantes como um tipo de dados Date:

```
Dim Today As Date
```

```
Dim StartTime As Date
```

```
Const FirstDay As Date = #1/1/2010
```

```
Const Noon = #12:00:00#
```

No VBA, coloque datas e horários entre duas marcas de cerquilha, conforme mostrado nos exemplos anteriores.

Variáveis Date exibem datas de acordo com o formato de data do seu **BRE-SE**

sistema e elas exibem os horários conforme o formato de horário do **LEM**

sistema (seja formatação de 12 ou 24 horas). O Registro do Windows armazena essas configurações e você pode modificá-las através da caixa de diálogo Opções Regionais e de Idioma no Painel de Controle do Windows. Portanto, o formato de data ou horário exibido pelo VBA pode variar, dependendo das configurações do sistema no qual o aplicativo está rodando.

No entanto, ao escrever código VBA, você precisa usar um dos formatos de data norte-americano (tal como mm/dd/yyyy). Assim, a declara-

ção a seguir designa um dia de outubro à variável MyDate (mesmo se o seu sistema estiver configurado para usar dd/mm/aaaa para datas):
MyDate = #10/11/2009#

Quando você exibe a variável (com a função MsgBox, por exemplo), VBA mostra MyDate usando as suas configurações de sistema. Portanto, se o seu sistema usar o formato de data dd/mm/aaaa, MyDate exibirá como 11/10/2009.

ExcelVBABokks.indb 108

15/02/2013 13:59:29

Capítulo 7: Elementos Essenciais da Linguagem VBA 109

Usando Declarações de Atribuição

Uma *declaração de atribuição* é uma declaração VBA que atribui o resultado de uma expressão a uma variável ou a um objeto. O sistema de Ajuda do Excel define o termo expressão como ... uma combinação de palavras-chave, operadores, variáveis e

constantes que produzem uma string, número ou objeto. Uma expressão pode ser usada para executar um cálculo, manipular caracteres ou testar dados.

Eu não teria dito isso melhor, portanto nem ao menos tento.

Muito do seu trabalho em VBA envolve o desenvolvimento (ou depura-

ção) de expressões. Se você souber como criar fórmulas em Excel, não terá problemas para criar expressões. Com uma fórmula da planilha, o Excel exibe o resultado em uma célula. Por outro lado, uma expressão VBA pode ser atribuída a uma variável.

Exemplos de declaração de atribuição

Nos exemplos de declaração de atribuição a seguir, as expressões estão à direita do sinal de igual:

$x = 1$

$x = x + 1$

$x = (y \ 2) / (z \ 2)$

HouseCost = 375000

FileOpen = True

Range("TheYear").Value = 2012

Expressões podem ser tão complexas quanto você precisa que elas
DICA

sejam: Use o caractere de continuação de linha (um espaço seguido por um sublinhado) para facilitar a leitura de expressões mais longas.

Geralmente, expressões usam funções: funções integradas do VBA, funções de planilha do Excel ou funções que você desenvolve com VBA.

Eu discuto funções no Capítulo 9.

Sobre aquele sinal de igual

Como pode ser visto no exemplo anterior, VBA usa o sinal de igual como seu operador de atribuição. Provavelmente, você está habituado ExcelVBABokks.indb 109

15/02/2013 13:59:29

110 Parte III: Conceitos de Programação a usar um sinal de igual como um símbolo matemático de igualdade.

Portanto, uma atribuição de declaração como a seguinte pode levá-lo a erguer suas sobrancelhas:

$z = z + 1$

Como a variável z pode ser igual a si mesma mais 1? Resposta: Não pode. Nesse caso, a declaração de atribuição aumenta o valor de z por 1. Lembre-se apenas que uma atribuição usa o sinal de igual como um operador, não um símbolo de igualdade.

Operadores regulares

Os operadores exercem uma importante função em VBA. Além do operador de sinal de igual (discutido na seção anterior), o VBA oferece vários outros operadores. A Tabela 7-3 relaciona esses operadores, com os quais você está familiarizado pela sua experiência em fórmulas de planilha.

Tabela 7-3

Operadores do VBA

Função

Símbolo do Operador

Adição

+

Multiplicação

*

Divisão

/

Subtração

-

Exponenciação ^

Concatenação de string

&

Divisão de inteiro (o resultado é sempre um inteiro)

\

Módulo aritmético (retorna o resto de uma operação

Mod

de divisão)

O termo *concatenação* é conversa de programador para “juntar”. Assim, se você concatena strings, está combinando strings para formar uma **DE ESPECIALISTA**

O

nova e aperfeiçoada string.

P A P

Conforme mostrado na Tabela 7-4, VBA também oferece um conjunto completo de operadores lógicos. Para detalhes completos, consulte o sistema de Ajuda.

ExcelVBABokks.indb 110

15/02/2013 13:59:30

Capítulo 7: Elementos Essenciais da Linguagem VBA 111

Tabela 7-4

Operadores Lógicos do VBA

Operador

O que ele faz

Not

Executa uma negação lógica em uma expressão.

And

Executa uma combinação lógica em duas expressões.

Or

Executa uma separação em duas expressões.

XoR

Executa uma exclusão em duas expressões.

Eqv

Executa uma equivalência lógica em duas expressões.

Imp

Executa uma implicação lógica em duas expressões.

A ordem de prioridade para operadores em VBA é exatamente a mesma que em fórmulas do Excel. A exponenciação tem a maior prioridade. A multiplicação e a divisão vêm em seguida e, depois, vêm a adição e subtração. Você pode usar parênteses para alterar a ordem natural de prioridade, colocando entre parênteses o que vier antes de qualquer operador. Dê uma olhada nesse código: $x = 3$

$y = 2$

$z = x + 5 * y$

Quando este código é executado, qual é o valor de z? Se você

respondeu 13, vai receber uma medalha de ouro, provando que você entende o conceito de prioridade do operador. Se você respondeu 16, leia isto: o operador de multiplicação ($5 * y$) é executado primeiro e o seu resultado é somado com x. Se você respondeu alguma outra coisa diferente de 13

ou 16, eu não tenho comentários.

A propósito, eu nunca consigo lembrar como funciona a prioridade de operador, portanto, tenho a tendência de usar parênteses mesmo quando eles não são exigidos. Por exemplo, na vida real eu escreveria aquela última declaração de atribuição assim: $z = x + (5 * y)$

DICA

Não se acanhe em usar parênteses mesmo que eles não sejam exigidos — especialmente porque se você fizer isso o seu código ficará mais fácil de entender. O VBA não se importa se você usa parênteses extras.

ExcelVBABokks.indb 111

15/02/2013 13:59:30

112 Parte III: Conceitos de Programação *Trabalhando com Arrays*

A maioria das linguagens de programação suporta arrays (matrizes). Um *array* é um grupo de variáveis que compartilha um nome comum. Você faz referência a uma variável específica no array, usando o nome do array e um número de índice entre parênteses. Por exemplo, é possível definir um array de 12 strings variáveis para conter os nomes dos meses do ano.

Se você nomear o array como *MonthNames*, pode se referir ao primeiro elemento do array como *MonthNames(1)*, ao segundo elemento como *MonthNames(2)* e assim por diante.

Declarando arrays

Antes de poder usar um array, você *deve* declará-lo. Sem expectativas.

Diferente das variáveis normais, VBA é bem rígido quanto a esta regra.

Você declara um array com uma declaração *Dim* ou *Public*, exatamente como declara uma variável regular. No entanto, você precisa especificar sempre o número de elementos no array. Isso é

feito especificando o primeiro número de índice, a palavra chave To e o último número de índice — tudo entre parênteses. O exemplo a seguir mostra como declarar um array de 100 números inteiros: Dim MyArray(1 To 100) As Integer

Ao declarar um array, você pode escolher especificar apenas o índice superior. VBA supõe que 0 é o índice inferior. Portanto, ambas as declarações a seguir indicam o mesmo array de 101 elementos: Dim MyArray (0 To 100) As Integer

Dim MyArray (100) As Integer

Se você quer que o VBA aceite que 1 (ao invés de 0) é o índice inferior

DICA

em seus arrays, inclua a seguinte declaração na seção Declarações de seu módulo:

Option Base 1

Essa declaração força o VBA a usar 1 como o primeiro número de índice em arrays que só declaram o índice superior. Se essa declaração estiver presente, as seguintes declarações são idênticas, ambas declarando um array de 100 elementos: Dim MyArray (1 To 100) As Integer

Dim MyArray (100) As Integer

ExcelVBABokks.indb 112

15/02/2013 13:59:30

Capítulo 7: Elementos Essenciais da Linguagem VBA 113

Arrays multidimensionais

Os arrays criados nos exemplos anteriores são todos unidimensionais.

Pense em um array unidimensional como uma única linha de valores. Os arrays que você cria em VBA podem ter até 60 dimensões — ainda que raramente você precise de mais do que duas ou três em um array.

O

seguinte exemplo declara um array de 81 inteiros com duas dimensões: Dim MyArray (1 To 9, 1 To 9) As Integer

Você pode pensar nesse array como ocupando uma matriz de 9 x 9 – perfeito para armazenar todos os números em um enigma sudoku.

Para fazer referência a um elemento específico neste array, você precisa especificar dois números de índice (semelhante à sua “linha” e sua

“coluna” na matriz). O exemplo a seguir mostra como é possível atribuir um valor a um elemento neste array:

```
MyArray (3, 4) = 125
```

Esta declaração atribui um valor a um único elemento no array. Se você pensar no array em termos de uma matriz de 9 x 9, isso designa 125 ao elemento localizado na terceira fileira e quarta coluna da matriz.

Eis como declarar um array tridimensional com 1.000 elementos: Dim My3DArray (1 To 10, 1 To 10, 1 To 10) As Integer

Você pode pensar de um array tridimensional como um cubo. É muito difícil visualizar um array com mais de três dimensões. Lamento, mas ainda não consegui administrar a quarta dimensão e além.

Arrays dinâmicos

Você também pode criar arrays *dinâmicos*. Um array dinâmico não tem um número pré-ajustado de elementos. Declare um array dinâmico com um conjunto vazio de parênteses: Dim MyArray () As Integer

Antes de poder usar este array, você deve usar a declaração ReDim para informar ao VBA quantos elementos tem o array. Em geral, a quantidade de elementos no array é determinada enquanto o seu código está rodando. Você pode usar a declaração ReDim quantas vezes quiser, trocando o tamanho do array tanto quanto necessário. O

exemplo a seguir demonstra como alterar o número de elementos em um array dinâmico. O exemplo supõe que a variável NumElements (número de elementos) contenha uma variável que o seu código calculou.

ExcelVBABokks.indb 113

15/02/2013 13:59:30

114 Parte III: Conceitos de Programação ReDim MyArray (1 To

NumElements)

Quando você redimensiona um array usando ReDim, você apaga

CUIDADO!

quaisquer valores atualmente armazenados nos elementos do array. É

possível evitar destruir os valores antigos usando a palavra-chave Preserve. O seguinte exemplo mostra como é possível preservar os valores de um array quando você o redimensiona: ReDim Preserve MyArray(1 To NumElements)

Se no momento MyArray tiver dez elementos e você executar a declara-

ção anterior com NumElements igualando 12, os primeiros dez elementos permanecem intactos, e o array tem espaço para dois elementos adicionais (até o número contido na variável NumElements). Mas, se NumElements for igual a 7, os primeiros sete elementos são retidos e os três elementos restantes são descartados.

O assunto de arrays volta no Capítulo 10, quando discuto sobre loop.

Usando Labels (Etiquetas)

Em versões anteriores de BASIC, cada linha de código exigia um número de linha. Por exemplo, se você tivesse escrito um programa BASIC nos anos 70 (claro que usando suas calças boca de sino), ela seria parecida com algo assim: 010: LET X=5

020: LET Y=3

030: LET Z=X*Y

040: PRINT Z

050: END

VBA permite o uso de tais números de linha e permite até mesmo o uso **BRE-SE**

de rótulos. Tipicamente, você não usa um rótulo em cada linha, mas **LEM**

pode, ocasionalmente, precisar usar um rótulo. Por exemplo, insira um rótulo se usar uma declaração GoTo (que eu discuto no Capítulo 10).

Um rótulo deve começar com o primeiro caractere sem espaço em uma linha e terminar com dois pontos.

As informações neste capítulo ficam mais claras à medida que você lê os capítulos subsequentes. Se quiser descobrir mais sobre os elementos da linguagem VBA, eu indico o sistema de Ajuda VBA. Você pode encontrar tantos detalhes quanto precise ou se preocupe em saber.

ExcelVBABokks.indb 114

15/02/2013 13:59:30

Capítulo 8

Trabalhando com

Objetos Range



Neste Capítulo

- ☐ Descobrindo por que objetos Range são tão importantes
- ☐ Como entender as várias maneiras de fazer referência a faixas
- ☐ Descobrindo algumas das propriedades mais úteis do objeto Range
- ☐ Descobrindo alguns dos métodos mais úteis do objeto Range



N este capítulo, eu vou um pouco mais fundo nos calabouços do Excel e olho atentamente os objetos Range. O Excel é totalmente voltado às células e o objeto Range é um contêiner para células. Por que você precisa saber tanto sobre objetos Range? Porque, muito do trabalho de programação que você faz no Excel, tem como foco os objetos Range. Pode me agradecer mais tarde.

Uma Revisão Rápida

O *objeto* Range representa uma faixa contida em um objeto Worksheet (pasta de trabalho). Objetos Range, como todos os outros objetos, têm propriedades (as quais você pode examinar e alterar) e métodos (que executam ações no objeto).

Um objeto Range pode ser tão pequeno quanto uma única célula (por exemplo, B4) ou tão grande quanto cada uma das 17.179.869.184 células em uma planilha (A1:XFD1048576).

Ao se referir a um objeto Range, o endereço é sempre rodeado por aspas duplas, assim:

Range("A1:C5")

ExcelVBABokks.indb 115

15/02/2013 13:59:30

116 Parte III: Conceitos de Programação Se a faixa consistir de uma célula, você ainda precisa das aspas: Range("K9")

Se acontecer da faixa ter um nome (criado usando Fórmulas⇒Nomes Definidos⇒Definir Nome), você pode usar uma expressão como esta: Range("PriceList")

A menos que você diga ao Excel o contrário, qualificando a faixa de referência, ele imagina que você está se referindo a uma faixa na **CUIDADO!**

planilha ativa. Se alguma outra coisa que não seja uma planilha estiver ativa (tal como um gráfico), a faixa de referência falha e a sua macro exibe uma mensagem de erro.

Conforme mostrado no seguinte exemplo, é possível referenciar uma faixa fora da planilha ativa, qualificando a faixa de referência com um nome de planilha a partir de uma pasta de trabalho ativa: Worksheets("Sheet1").Range("A1:C5")

Se você precisar fazer referência a uma faixa em uma pasta de trabalho diferente (isto é, qualquer pasta de trabalho que não aquela ativa), é possível usar uma declaração como esta: Workbooks("Budget.xlsl").Worksheets("Sheet1"). _

Range("A1:C5")

Um objeto Range pode consistir de uma ou mais linhas ou colunas inteiras. Você pode fazer referência a toda uma linha (nesse caso, linha 3), usando uma sintaxe como esta:

Range("3:3")

É possível fazer referência a uma coluna inteira (coluna 4 neste

exemplo) assim:

Range(“D:D”)

No Excel, você seleciona faixas intercaladas, mantendo pressionada a tecla Ctrl enquanto seleciona várias faixas com o seu mouse. A Figura 8-1 mostra uma seleção de faixas intercaladas. Você não ficaria surpreso ao saber que VBA também lhe permite trabalhar com faixas intercaladas (não contínuas). A seguinte expressão refere-se a uma faixa não contínua de duas áreas. Observe que uma vírgula separa as duas áreas.

Range(“A1:B8,D9:G16”)

Esteja ciente de que alguns métodos e propriedades causam danos a **CUIDADO!**

faixas não contínuas. Você pode ter que processar cada área separadamente, usando um loop.

ExcelVBABokks.indb 116

15/02/2013 13:59:31

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							

Figura 8-1:

Uma
seleção
de faixas
intercaladas.

Outras Maneiras de Fazer Referência

a uma Faixa

Quanto mais você trabalha com VBA, mais percebe que ele é uma linguagem muito bem concebida e, geralmente, bastante lógica (apesar do que você pode estar pensando agora). Com frequência, o VBA oferece múltiplas maneiras de executar uma ação. Você pode escolher o método mais adequado ao seu problema. Esta seção discute algumas das formas de fazer referência a uma faixa.

Este capítulo mal toca a superfície das propriedades e métodos do **BRE-SE**

objeto Range. Ao trabalhar com VBA, provavelmente você precisará **LEM**

acessar outras propriedades e métodos. O sistema de Ajuda é o melhor lugar para descobrir a respeito delas, mas também é uma boa ideia gravar suas ações e examinar o código que o Excel gerar.

A propriedade Cells

Ao invés de usar a palavra-chave VBA Range, é possível fazer referência a uma faixa através da propriedade Cells.

Observe que eu escrevi *propriedade Cells* e não *objeto Cells* ou mesmo *coleção Cells*. Ainda que Cells possa parecer com um objeto (ou com **DE ESPECIALISTA O P**

uma coleção), na verdade não é. Ao contrário, Cell é uma propriedade **A P**

que o VBA avalia. Depois, o VBA retorna um objeto (mais especificamente, um objeto Range). Se isso parecer estranho, não se preocupe.

Até a Microsoft parece estar confusa quanto a essa questão. Em algumas versões anteriores de Excel, a propriedade Cells era conhecida como o método Cells. Independente do que ela é, apenas entenda que Cells é uma maneira jeitosa de fazer referência a uma faixa.

ExcelVBABokks.indb 117

15/02/2013 13:59:31

118 Parte III: Conceitos de Programação A propriedade Cells toma dois argumentos: linha e coluna. Esses dois argumentos são números, ainda que normalmente nos referimos a colunas usando letras. Por exemplo, a seguinte expressão refere-se à célula C2 em Sheet2: Worksheets(“Sheet2”).Cells(2, 3)

Também é possível usar a propriedade Cells para fazer referência a uma faixa de múltiplas células. O seguinte exemplo demonstra a sintaxe que você usa:

```
Range(Cells(1, 1), Cells(10, 10))
```

Esta expressão refere-se a uma faixa de célula 100, que se estende da célula A1 (linha 1, coluna 1) à célula J10 (linha 10, coluna 10).

As duas declarações seguintes produzem o mesmo resultado: elas entram com um valor de 99 em uma faixa de células 10 por 10. Mais especificamente, essas declarações configuram a propriedade Value do objeto Range: Range(“A1:J10”).Value = 99

```
Range(Cells(1, 1), Cells(10, 10)).Value = 99
```

A vantagem de usar o método Cells para referenciar faixas torna-se
DICA

clara quando você usa variáveis ao invés de números como argumentos de Cells. E as coisas começam, de fato, a aparecer quando você entende sobre loop, que discuto no Capítulo 10.

A propriedade Offset

A propriedade Offset oferece outra forma jeitosa de fazer referência a células. Essa propriedade, a qual opera em um objeto Range e retorna outro objeto Range, permite que você faça referência a uma célula que está em dado número de linhas e colunas distante de outra célula.

Como a propriedade Cells, a propriedade Offset toma dois argumentos.

O primeiro argumento representa o número de linhas a deslocar; o segundo representa o número de colunas a deslocar.

A seguinte expressão refere-se a uma célula localizada a uma linha abaixo e duas colunas à direita da célula A1. Em outras palavras, essa referência à célula geralmente é conhecida como C2:

Range("A1").Offset(1, 2)

ExcelVBABokks.indb 118

15/02/2013 13:59:32

Capítulo 8: Trabalhando com Objetos Range 119

O método Offset também pode usar argumentos negativos. Um offset de linha *negativo* refere-se a uma linha acima da faixa. Um offset de coluna negativo refere-se a uma coluna à esquerda da faixa. O seguinte exemplo refere-se à célula A1: Range("C2").Offset(-1, -2)

E, como você poderia esperar, é possível usar 0 como um ou ambos os argumentos para Offset. A seguinte expressão refere-se à célula A1: Range("A1").Offset(0, 0)

O método Offset é mais útil quando você usa variáveis ao invés de **DICA**

valores nos argumentos. No Capítulo 10 apresento alguns exemplos demonstrando isso.

Fazendo referência a colunas e linhas inteiras

Se você precisa referenciar uma faixa que consiste de uma ou mais colunas inteiras, pode usar uma expressão como a seguinte:

Columns("A:C")

E, para fazer referência a uma ou mais linhas inteiras, use uma expressão assim:

Rows("1:5")

Algumas Propriedades úteis

do Objeto Range

Um objeto Range tem dúzias de propriedades. Você pode escrever programas VBA sem parar pelos próximos dez anos e nunca usá-las todas.

Descrevo rapidamente algumas das propriedades Range usadas mais frequentemente. Para mais detalhes, consulte o sistema de Ajuda no VBE.

Algumas propriedades Range são *apenas de leitura*, significando que **BRE-SE**

you pode fazer com que o seu código olhe para os seus valores, mas **LEM**

não pode fazer com que ele os altere (“olhe, mas não toque”). Por exemplo, cada objeto Range tem uma propriedade Address que pode conter o endereço da faixa). Você pode acessar essa propriedade, mas não pode alterá-la – ela é apenas de leitura.

A propósito, os exemplos a seguir são tipicamente declarações ao invés de procedimentos completos. Se você quiser experimentar uma delas (o que deveria fazer), crie um procedimento Sub para fazê-lo. Da mesma forma, muitas dessas declarações só funcionam adequadamente na planilha ativa.

ExcelVBABokks.indb 119

15/02/2013 13:59:32

120 Parte III: Conceitos de Programação DE ESPECIALISTA

O P A P

Atribuindo os valores de um intervalo

de células a uma variável

Eu não estava sendo totalmente sincero Dim x As Variant

quando escrevi “você pode ler a propriedade Valor apenas em um objeto Range de cé-

x = Range(“A1:C3”).Value

lula única”. Na verdade, você pode atribuir Depois, você pode tratar a

variável x como os valores de um intervalo com múltiplas se ela fosse um array. Essa declaração, por células a uma variável, desde que a variável exemplo, retorna o valor na célula B1: seja uma variante. Isso porque uma variante pode agir como um array. Eis um exemplo:

```
MsgBox x(1, 2)
```

A propriedade Value

A propriedade Value representa o valor contido em uma célula. Ela é uma propriedade de leitura e escrita, portanto, o seu código VBA pode ler ou alterar o valor.

A seguinte declaração exibe uma caixa de mensagem que exibe o valor da célula A1 em Sheet1:

```
MsgBox Worksheets("Sheet1").Range("A1").Value
```

No entanto você pode ler a propriedade Value apenas de um objeto Range de célula única. Por exemplo, a seguinte declaração gera um erro: `MsgBox Worksheets("Sheet1").Range("A1:C3").Value` Mas, você também pode usar a propriedade Value com uma faixa de qualquer tamanho. A seguinte declaração entra com o número 123 em cada célula em uma faixa:

```
Worksheets("Sheet1").Range("A1:C3").Value = 123
```

Value é a propriedade padrão em um objeto Range. Em outras palavras, **DICA**

se você omitir uma propriedade para o objeto, o Excel usa a sua propriedade Value. As duas declarações a seguir atribuem valor 75 para a célula A1 em Sheet1:

```
Worksheets("Sheet1").Range("A1").Value = 75
```

```
Worksheets("Sheet1").Range("A1") = 75
```

ExcelVBABokks.indb 120

15/02/2013 13:59:33

Capítulo 8: Trabalhando com Objetos Range 121

A propriedade Text

A propriedade `Text` retorna uma string que representa o texto conforme exibido em uma célula – o valor formatado. A propriedade `Text` é apenas de leitura. Por exemplo, suponha que a célula A1 contenha o valor 12.3 e esteja formatada para exibir dois decimais e um cifrão (\$12.30). A seguinte declaração exibe uma mensagem: `MsgBox Worksheets("Sheet1").Range("A1").Text`

Porém, a próxima declaração exibe uma caixa de mensagem contendo 12.3:

```
MsgBox Worksheets("Sheet1").Range("A1").Value
```

Se a célula contém uma fórmula, a propriedade `Text` retorna o resultado da fórmula. Se uma célula contiver texto, então a propriedade `Text` e a propriedade `Value` sempre retornarão a mesma coisa, pois texto (diferente de um número) não pode ser formatado para ser exibido diferentemente.

A propriedade Count

A propriedade `Count` retorna quantidade de células de uma faixa. Ela conta todas as células, não apenas as células que não estão em branco.

`Count` é uma propriedade apenas de leitura (pense nisso por um segundo e entenderá o motivo). A seguinte declaração encaminha uma propriedade `Count` da faixa e exibe o resultado (9) em uma caixa de mensagem: `MsgBox Range("A1:C3").Count`

As propriedades Column e Row

A propriedade `Column` retorna o número da coluna de uma faixa de célula única. Ela é amiga íntima da propriedade `Row`, que retorna o número da linha de uma faixa de célula única. Ambas as propriedades são apenas de leitura. Por exemplo, o resultado da seguinte declaração é 6, pois a célula F3 está na sexta coluna: `MsgBox Sheets("Sheet1").Range("F3").Column`

A próxima expressão resulta em 3, porque a célula F3 está na terceira linha: `MsgBox Sheets("Sheet1").Range("F3").Row`



122 Parte III: Conceitos de Programação

Se o objeto Range consistir de mais de uma célula, a propriedade Column **BRE-SE**

retorna o número da primeira coluna na faixa, e a propriedade Row **LEM**

retorna o número da primeira linha na faixa.

Não confunda as propriedades Column e Row com as propriedades Columns e Rows (discutidas anteriormente neste capítulo). As propriedades **CUIDADO!**

Column e Row retornam um único valor. As propriedades

Columns e Rows retornam um objeto Range. Que diferença um “s” faz.

A propriedade Address

Address uma propriedade apenas de leitura, exibe o endereço de célula para um objeto Range com notação absoluta (um cifrão antes da letra da coluna e antes do número da linha). A declaração a seguir exibe a caixa de mensagem mostrada na Figura 8-2.

```
MsgBox Range(Cells(1, 1), Cells(5, 5)).Address
```

Figura 8-2:

Esta caixa

de mensagem exibe a

propriedade

Address de

uma faixa de

1 por 5.

A propriedade HasFormula

A propriedade HasFormula (que é apenas de leitura) retorna True se a faixa de célula única contiver uma fórmula. Ela retorna False se a célula não tiver uma fórmula. Se a faixa consistir em mais de uma célula, o VBA só retorna True se todas as células da faixa contiver uma fórmula ou retorna False se nenhuma célula da faixa contiver uma fórmula. A propriedade retorna um Null (nulo) caso se apenas uma parte das células contenha fórmulas na faixa. Null é uma espécie de terra de ninguém: a faixa contém uma mistura de fórmulas e valores.

Você precisa, ter cuidado ao trabalhar com propriedades que podem **CUIDADO!**

retornar Null. Mais especificamente, o único tipo de dados que pode lidar com Null é Variant.

Por exemplo, suponha que a célula A1 contenha um valor e a célula A2 contenha uma fórmula. As seguintes declarações geram um erro, porque a faixa não possui fórmulas em todas ou em nenhuma de suas células:

ExcelVBABokks.indb 122

15/02/2013 13:59:33

Capítulo 8: Trabalhando com Objetos Range 123

Dim FormulaTest As Boolean

FormulaTest = Range("A1:A2").HasFormula

O tipo de dados Boolean só pode lidar com True ou False. Null leva-o a reclamar e responder com uma mensagem de erro. Para corrigir esse tipo de situação, a melhor coisa a fazer é garantir que a variável FormulaTest seja declarada como Variant, ao invés de Boolean. O exemplo a seguir usa convenientemente a função TypeName (digitar nome) do VBA (juntamente com uma declaração If-Then) para determinar o tipo de dados da variável FormulaTest. Se a faixa misturar fórmulas e valores, a caixa de mensagem exibe *Mixed!*

(misturado).

Dim FormulaTest As Variant

FormulaTest = Range("A1:A2").HasFormula

If TypeName(FormulaTest) = "Null" Then MsgBox "Mixed!"

A propriedade Font

Como anteriormente mencionado neste capítulo (veja “A propriedade Cells”), uma propriedade pode retornar um objeto. A propriedade Font de um objeto Range é um outro exemplo daquele conceito em opera-

ção. A propriedade Font retorna um objeto Font.

Como seria esperado, um objeto Font tem muitas propriedades acessíveis.

Para alterar alguns aspecto, de uma fonte da faixa, primeiro você precisa acessar o objeto Font da faixa e depois, manipular as propriedades daquele objeto. Isso pode ser confuso, mas talvez o exemplo a seguir ajude.

A seguinte declaração usa a propriedade Font do objeto Range para retornar um objeto Font. Depois, a propriedade Bold do objeto Font é configurada para True. Em inglês puro, isso faz o conteúdo da célula ser exibido em negrito: Range("A1").Font.Bold = True

A verdade é que você não precisa, de fato, saber que está trabalhando com um objeto Font que está contido em um objeto Range. Desde que use a sintaxe apropriada, ela funcionará bem. Com frequência, gravar as suas ações enquanto você grava uma macro o deixará informado sobre tudo o que precisa saber sobre a sintaxe certa.

Para mais informações sobre gravação de macros, veja o Capítulo 6.

A propriedade Interior

Eis um outro exemplo de uma propriedade que retorna um objeto. A propriedade Interior do objeto Range retorna um objeto Interior (nome estranho, mas é assim que ele é chamado). Esse tipo de objeto de referência funciona da mesma maneira que a propriedade Font (a qual descrevi na seção anterior).

124 Parte III: Conceitos de Programação Por exemplo, a declaração a seguir muda a propriedade Color do objeto Interior contido no objeto Range: `Range("A1").Interior.Color = 8421504`

Em outras palavras, esta declaração muda o fundo da célula para cinza médio. O que é isso? Você não sabia que 8421504 é cinza médio? Devido a alguma perspicácia no maravilhoso mundo das cores do Excel, leia o artigo “Um rápido e sujo resumo sobre as cores” na próxima página.

A propriedade Formula

A propriedade Formula representa a fórmula em uma célula. Essa é uma propriedade de leitura e escrita, portanto, você pode acessá-la para inserir uma fórmula em uma célula. Por exemplo, a seguinte declaração entra com uma fórmula SUM na célula A13: `Range("A13").Formula = "=SUM(A1:A12)"`

Veja que a fórmula é uma string de texto e está entre aspas.

Se a fórmula, tiver ela mesma, aspas, as coisas ficam um pouco mais ardilosas. Por exemplo, digamos que você quer inserir esta fórmula usando VBA:

```
=SUM(A1:A12)&" Stores"
```

Esta fórmula exibe um valor, seguido pela palavra *Stores*. Para tornar esta fórmula aceitável, é preciso substituir cada aspa na fórmula por duas aspas. Caso contrário, o VBA ficará confuso e responderá com o aviso de um erro de sintaxe (porque há!). Assim, eis uma declaração que entrará com uma fórmula que contém aspas:

```
Range("A13").Formula = "=SUM(A1:A12)&"" Stores"""
```

A propósito, é possível acessar a propriedade Formula de uma célula mesmo que a célula não tenha uma fórmula. Se uma célula não tem fórmula, a propriedade Formula retorna o mesmo que sua propriedade Value.

Se você precisar saber quando uma célula tem fórmula, use a propriedade HasFormula.

Esteja atento ao fato de que o VBA “fala” inglês norte-americano. Isso **BRE-SE**

significa que para colocar uma fórmula em uma célula, você precisa **LEM**

usar a sintaxe norte-americana. Para usar a sua própria sintaxe de fórmula local em VBA, verifique a propriedade `FormulaLocal`.

ExcelVBABokks.indb 124

15/02/2013 13:59:34

Capítulo 8: Trabalhando com Objetos Range 125

Um rápido e sujo resumo sobre as cores

Antes do Excel 2007, a Microsoft tentou nos convencer de que 56 cores eram suficientes para uma planilha básica. Mas, as coisas mudaram e podemos usar mais de 16 milhões de cores em uma pasta de trabalho, 16.777.216 cores, para ser exato.

Muitos objetos têm uma propriedade `Color` e essa propriedade aceita valores de cor que variam de 0 a 16777215. Ninguém pode se lembrar de tantos valores de cor, assim (felizmente) há uma maneira mais fácil de especificar cores: use a função RGB (Red-Green-Blue) do VBA. Essa função tem a vantagem de que qualquer dessas 16 milhões de cores pode ser representada por vários níveis de vermelho, verde e azul. Os três argumentos na função RGB correspondem aos componentes das cores vermelho, verde e azul, e cada um desses argumentos pode variar de 0 a 255.

Observe que $256 \times 256 = 16.777.216$ — que por acaso é a quantidade de cores. Você não adora quando a matemática funciona?

A seguir, estão alguns exemplos que usam a função RGB para mudar a cor de fundo de uma célula: `Range("A1").Interior.Color = RGB(9, 0, 0) 'black`

`Range("A1").Interior.Color = RGB(255, 0, 0) 'pure red`

`Range("A1").Interior.Color = RGB(0, 0, 255) 'pure blue`

`Range("A1").Interior.Color = RGB(200, 89, 18) 'orangy-brown`

`Range("A1").Interior.Color = RGB(128, 126, 128) 'middle gray`

Se você quiser usar cores padrão, optar por usar uma das cores

constantes integradas: vbBlack, vbRed, vbGreen, vbYellow, vbBlue, vbMagenta, vbCyan ou vbWhite. Por exemplo, a seguinte declaração torna amarela a célula A1: Range("A1").Interior.Color = vbYellow

O Excel 2007 também introduziu “cores de tema”. Essas são cores que aparecem quando você usa o controle de cor, tal como o controle Fill Color (preencher com cor) no grupo Font da guia Página Inicial.

Experimente gravar uma macro enquanto muda cores e você receberá algo assim: Range("A1").Interior.ThemeColor = xlThemeColorAccent4

Range("A1").Interior.TintAndShade = 0.399975585192419

É, mais duas propriedades referentes a cor para lidar. Aqui, temos uma cor de tema (a cor básica, especificada como uma constante integrada), mais um valor “tint and shade” (matiz e tonalidade) que representa quão escura ou clara é a cor. Valores TintAndShade variam de -1.0 a +1.0. Valores positivos da propriedade TintAndShade tornam a cor mais clara e valores negativos tornam a cor mais escura. Quando você ajusta uma cor usando a propriedade ThemeColor, a cor mudará se você aplicar um documento de tema diferente (usando o comando Layout da Página⇒Temas⇒Temas).

A propriedade NumberFormat

A propriedade NumberFormat representa o formato do número (expresso como uma string de texto) do objeto Range. Essa é uma propriedade de leitura e escrita, assim, o seu código VBA pode mudar o formato do ExcelVBABokks.indb 125

15/02/2013 13:59:34

126 Parte III: Conceitos de Programação número. A declaração a seguir altera o formato do número da coluna A para porcentagem com duas casas decimais: Columns("A:A").NumberFormat = "0.00%"

Siga as etapas a seguir para ver uma relação de outros formatos de número. Melhor ainda, ligue o gravador de macro enquanto fizer isso.

1. Ative uma planilha.

2. Acesse a caixa de diálogo Formatar Células, pressionando Ctrl + 1.

3. Selecione a guia Número.

4. Selecione a categoria Personalizado para ver e aplicar algumas strings adicionais de formato de número.

Alguns Métodos úteis do Objeto Range

Como você sabe, um método VBA executa uma ação. Um objeto Range tem dezenas de métodos, mas, de novo, você não precisa da maioria deles.

Nesta seção, indico alguns dos métodos mais usados do objeto Range.

O método Select

Use o método Select para selecionar uma faixa de células. A seguinte declaração seleciona uma faixa na planilha ativa:

```
Range("A1:C12").Select
```

Antes de selecionar uma faixa, geralmente é uma boa ideia usar uma **CUIDADO!**

declaração adicional para garantir que a planilha certa esteja ativa. Por exemplo, se Sheet1 contém a faixa que você deseja selecionar, use as seguintes declarações para selecionar a faixa:

```
Sheets("Sheet1").Activate
```

```
Range("A1:C12").Select
```

Ao contrário do que você pode esperar, a seguinte declaração gera um erro se Sheet1 não for a planilha ativa. Em outras palavras, você deve usar duas declarações ao invés de apenas uma: uma para ativar a planilha e outra para selecionar a faixa.

```
Sheets("Sheet1").Range("A1:C12").Select
```

DICA

Se você usar o método GoTo do objeto Application para selecionar uma faixa, pode esquecer sobre selecionar primeiro a planilha certa. Essa declaração ativa Sheet1 e depois seleciona a faixa:
ExcelVBABokks.indb 126

15/02/2013 13:59:35

Application.Goto Sheets(“Sheet1”).Range(“A1:C12”)

O método GoTo é o equivalente em VBA à função da tecla F5 no Excel, que exhibe a caixa de diálogo Ir para.

Os métodos Copy e Paste

Você pode executar as operações de copiar e colar em VBA usando os métodos Copy e Paste. Note que dois objetos diferentes entram em cena. O método Copy é aplicável ao objeto Range, mas o método Paste aplica-se ao objeto Worksheet. Na verdade, isso faz sentido: você copia uma faixa e a cola em uma planilha.

Esta macro curta (cortesia do gravador de macro) copia a faixa A1:A12

e a cola na mesma planilha, começando na célula C1:

```
Sub CopyRange()
```

```
Range(“A1:A12”).Select
```

```
Selection.Copy
```

```
Range(“C1”).Select
```

```
ActiveSheet.Paste
```

```
End Sub
```

Observe que no exemplo anterior, o objeto ActiveSheet (planilha ativa) **DICA**

é usado com o método Paste. Essa é uma versão especial do objeto Worksheet que se refere à planilha ativa no momento. Veja ainda que a macro seleciona a faixa antes de copiá-la. No entanto, você não tem que selecionar uma faixa antes de fazer alguma coisa com ela. Na verdade, o seguinte procedimento consegue realizar a mesma tarefa que o exemplo anterior, usando uma única declaração: Sub CopyRange1()

```
Range(“A1:A12”).Copy Range(“C1”)
```

```
End Sub
```

Este procedimento tem a vantagem de que o método Copiar pode usar um argumento que corresponde à faixa de destino para a operação de

cópia.

O método Clear

O método Clear (limpar) apaga o conteúdo de uma faixa, incluindo toda a formatação de célula. Por exemplo, se você quiser apagar tudo na coluna D, a seguinte declaração faz o trabalho:

Columns("D:D").Clear

ExcelVBABokks.indb 127

15/02/2013 13:59:35

128 Parte III: Conceitos de Programação Você também deve conhecer dois métodos correlatos. O método

ClearContents, que apaga o conteúdo de uma faixa, mas deixa a formatação intacta. E o método ClearFormats, que apaga a formatação na faixa, mas não o conteúdo da célula.

O método Delete

Limpar uma faixa é diferente de deletar ou excluir uma faixa. Quando você *deleta* (ou *exclui*) uma faixa, o Excel desloca as células restantes para preencher a faixa excluída.

O exemplo a seguir usa o método Delete para excluir a linha 6: Rows("6:6").Delete

Quando você exclui uma faixa que não é uma linha ou coluna inteira, o Excel precisa ser informado sobre como deslocar as células (para ver como isso funciona, experimente o comando Página Inicial⇒Células⇒Excluir).

A seguinte declaração exclui uma faixa e preenche o espaço resultante, com as outras células à sua esquerda:

Range("C6:C10").Delete xlToLeft

O método Delete usa um argumento que indica como o Excel deveria trocar as células restantes. Neste caso, eu uso uma constante integrada (xlToLeft) para o argumento. Eu também poderia usar xlUp, uma outra constante nomeada.

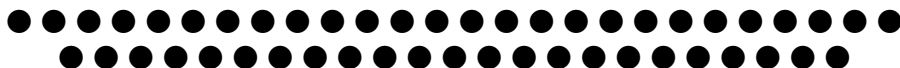
ExcelVBABokks.indb 128

15/02/2013 13:59:35

Capítulo 9

Usando VBA e Funções

de Planilha



Neste Capítulo

- ☐ Como usar funções para tornar suas expressões VBA mais poderosas
- ☐ Como usar funções VBA integradas
- ☐ Como usar funções de planilha do Excel em seu código VBA
- ☐ Como escrever funções personalizadas



Nos capítulos anteriores, eu mencionei o fato de que você pode usar funções em suas expressões VBA. Há três sabores de fun-

ções: as internas do VBA, as do Excel e outras funções escritas em VBA.

Neste capítulo, ofereço uma explicação completa sobre isso. As funções podem fazer o seu código VBA executar algumas façanhas poderosas, exigindo pouco ou nenhum esforço de programação. Se gostou da ideia, este capítulo é para você.

O que É uma Função?

Exceto por algumas pessoas que pensam que Excel é um processador de palavras, todos os usuários de Excel usam funções de planilha em suas fórmulas. A função mais comum de planilha é a função SUM, e você tem centenas de outras à sua disposição.

Essencialmente, uma *função* executa um cálculo e retorna um único valor. Claro que a função SUM retorna a soma de uma faixa de valores.

O mesmo se refere às funções usadas em suas expressões VBA: cada função faz isso e retorna um único valor.

As funções que você usa em VBA podem vir a partir de três fontes: 5
Funções integradas fornecidas pelo VBA

5 Funções de planilha fornecidas pelo Excel

5 Funções personalizadas que você (ou alguém) escreve, usando VBA
ExcelVBABokks.indb 129

15/02/2013 13:59:36

130 Parte III: Conceitos de Programação O resto deste capítulo esclarece as diferenças (espero) e o convence da importância de usar funções em seu código VBA.

Usando Funções VBA Integradas

O VBA oferece diversas funções integradas. Algumas dessas funções usam argumentos e outras não.

Exemplo de função VBA

Nesta seção, eu apresento alguns exemplos sobre o uso de funções VBA no código. Em muitos desses exemplos, uso a função MsgBox para exibir um valor em uma caixa de mensagem. Sim, MsgBox é uma função VBA — bem incomum, contudo, uma função. Essa útil função exibe uma mensagem em uma caixa de diálogo pop-up. Para mais detalhes sobre a função MsgBox, veja o Capítulo 15.

Uma pasta de trabalho contendo todos os exemplos está disponível no Web site deste livro.

Exibindo a data ou horário do sistema

O primeiro exemplo usa a função Date do VBA para exibir a data atual do sistema em uma caixa de mensagem:

```
Sub ShowDate()
```

```
MsgBox Date
```

```
End Sub
```

Veja que a função Date não usa um argumento. Diferente das funções

de planilha, uma função VBA sem argumento não exige um conjunto de parênteses vazios. Na verdade, se você digitar um conjunto de parênteses vazios, o VBE os removerá prontamente.

Para obter o horário do sistema, use a função Time. E, se você quiser mais, use a função Now para retornar a data e o horário.

Como encontrar a extensão de uma string

O procedimento a seguir usa a função VBA Len (length = comprimento), que retorna o comprimento de uma string de texto. A função Len toma um argumento: a string. Quando você executa esse procedimento, o resultado exibido na caixa de mensagem é 11, pois o argumento tem 11

caracteres.

ExcelVBABokks.indb 130

15/02/2013 13:59:36

Capítulo 9: Usando VBA e Funções de Planilha 131

```
Sub GetLength()
```

```
Dim MyString As String
```

```
Dim StringLength As Integer
```

```
MyString = "Hello World"
```

```
StringLength = Len(MyString)
```

```
MsgBox StringLength
```

```
End Sub
```

O Excel também tem uma função Len, que você pode usar em suas planilhas de fórmulas. A versão do Excel e a função VBA funcionam da mesma maneira.

Exibindo a parte inteira de um número

O seguinte procedimento usa a função Fix (corrigir), a qual retorna a parte *inteira* de um valor — o valor sem quaisquer casas decimais: Sub

```
GetIntegerPart()
```

```
Dim MyValue As Double
```

```
Dim IntValue As Integer
```

```
MyValue = 123.456
```

```
IntValue = Fix(MyValue)
```

```
MsgBox IntValue
```

```
End Sub
```

Neste caso, a caixa de mensagem exibe 123.

BRE-SE

VBA tem uma função semelhante, chamada Int. A diferença entre Int e LEM

Fix é como cada uma lida com números negativos. Trata-se de uma diferença sutil, mas, às vezes, é importante.

5 Int retorna o primeiro inteiro negativo que é menor ou igual ao argumento. Fix (-123.456) retorna -124.

5 Fix retorna o primeiro inteiro negativo que é maior ou igual ao argumento. Fix (-123.456) retorna -123.

Determinando o tamanho de um arquivo

O procedimento Sub a seguir exibe o tamanho, em bytes, de um arquivo Excel executável. Ele encontra o seu valor usando a função FileLen (extensão de arquivo).

```
Sub GetFileSize()
```

```
Dim TheFile As String
```

```
TheFile = "C:\Program Files\Microsoft Office\
```

```
Office14\Excel.exe"
```


MsgBox FileLen(TheFile)

End Sub

Veja que essa rotina *restringe* os códigos do nome de arquivo (isto é, declara explicitamente o caminho). Em geral, essa não é uma boa ideia. O

arquivo poderia não estar no drive C, ou a pasta do Excel pode ter um nome diferente. A seguinte declaração mostra uma abordagem melhor: ExcelVBABokks.indb 131

15/02/2013 13:59:37

132 Parte III: Conceitos de Programação TheFile = Application.Path & “\EXCEL.EXE”

Path (caminho) é uma propriedade do objeto Application. Ela simplesmente retorna o nome da pasta onde o aplicativo (isto é, o Excel) está instalado (sem uma barra invertida ao final).

Identificando o tipo de um objeto selecionado

O procedimento a seguir usa a função TypeName, que retorna o tipo da seleção na planilha (como uma string):

Sub ShowSelectionType()

Dim SelType As String

SelType = TypeName(Selection)

MsgBox

SelType

End Sub

O objeto poderia ser Range, Picture, Rectangle, ChartArea ou qualquer outro tipo de objeto que possa ser selecionado.

DICA

A função TypeName é muito versátil. Você também pode usar essa função para determinar o tipo de dados de uma variável.

Funções VBA que fazem mais do que

retornar um valor

Algumas funções VBA vão acima e além da obrigação. Ao invés de simplesmente retornar em um valor, essas funções têm alguns efeitos colaterais úteis. A Tabela 9-1 as relaciona.

Tabela 9-1

Funções com Benefícios Colaterais Úteis

Função

O que ela faz

MsgBox

Exibe uma caixa de diálogo útil contendo uma mensagem e botões.

A função retorna um código que identifica qual botão o usuário clica. Para mais detalhes, veja o Capítulo 15.

InputBox

Exibe uma simples caixa de diálogo que pede alguma entrada do usuário. A função retorna o que o usuário inserir na caixa de diá-

logo. Eu discuto isso no Capítulo 15.

Shell

Executa outro programa. A função retorna a ID (um identificador único) da tarefa do outro programa (ou um erro, se a função não puder iniciar o outro programa).

ExcelVBABokks.indb 132

15/02/2013 13:59:38

Capítulo 9: Usando VBA e Funções de Planilha 133

Descobrindo funções VBA

Como você descobre quais funções o VBA oferece? Boa pergunta. A melhor fonte é o sistema de Ajuda do Excel Visual Basic. Eu compilei uma lista parcial de funções, que compartilho com você na Tabela 9-2.

Omiti algumas das funções mais especializadas ou obscuras.

DICA

Para mais detalhes sobre uma função em especial, digite o nome da função em um módulo VBA, mova o cursor em qualquer lugar no texto e pressione F1.

Tabela 9-2 Funções mais úteis do VBA

Função

O Que Ela Faz

Abs

Retorna o valor absoluto de um número.

Array

Retorna uma variante contendo um array.

Asc

Converte o primeiro caractere de uma string ao seu valor ASCII.

Atn

Retorna o arco tangente de um número.

Choose

Retorna um valor de uma lista de itens.

Chr

Converte um valor ANSI a uma string.

Cos

Retorna o cosseno de um número.

CurDir

Retorna o caminho atual.

Date

Retorna a data atual do sistema.

DateAdd

Retorna uma data ao qual foi acrescentado um intervalo de

tempo especificado – por exemplo, um mês a partir de uma data em especial.

DateDiff

Retorna um inteiro mostrando o número de intervalos de tempo

especificados entre duas datas – por exemplo, o número de meses entre agora e o seu aniversário.

DatePart

Retorna um inteiro contendo a parte especificada de determinada data – por exemplo, um dia do ano da data.

DateSerial Converte uma data em um número em série.

DateValue Converte uma string em uma data.

Day

Retorna o dia do mês a partir de um valor de data.

Dir

Retorna o nome de um arquivo ou diretório que combina com um padrão.

Erl

Retorna o número da linha que causou um erro.

Err

Retorna o número de erro de uma condição de erro.

Error

Retorna a mensagem de erro que corresponde a um número de erro.

Exp

Retorna a base do logaritmo natural (e) elevado a uma potência.

(continua)

ExcelVBABokks.indb 133

15/02/2013 13:59:39

134 Parte III: Conceitos de Programação Tabela 9-2

(continuação)

Função

O Que Ela Faz

FileLen

Retorna o número de bytes de um arquivo.

Fix

Retorna a parte inteira de um número.

Format

Exibe uma expressão em um formato especial.

GetSetting Retorna o valor de um registro Windows.

Hex

Converte de decimal para hexadecimal.

Hour

Retorna a parte das horas de um horário.

InputBox

Exibe uma caixa para solicitar uma entrada pelo usuário.

InStr

Retorna a posição de uma string dentro de outra string.

Int

Retorna a parte inteira de um número.

IAmt

Retorna o pagamento de juros de uma anuidade ou empréstimo.

IsArray

Retorna Verdadeiro se uma variável for um array.

IsDate

Retorna Verdadeiro se uma expressão for uma data.

IsEmpty

Retorna Verdadeiro se uma variável não tiver sido inicializada.

IsError

Retorna Verdadeiro se uma expressão for um valor de erro.

IsMissing Retorna Verdadeiro se um argumento opcional não foi passado a um procedimento.

IsNull

Retorna Verdadeiro se uma expressão não contém dados válidos.

IsNumeric Retorna Verdadeiro se uma expressão pode ser avaliada como um número.

IsObject

Retorna Verdadeiro se uma expressão referenciar um objeto OLE Automation.

LBound

Retorna o menor subscrito para a dimensão de um array.

LCase

Retorna uma string convertida para minúsculas.

Left

Retorna uma string a partir do número de caracteres especificados à esquerda dessa string.

Len

Retorna o número de caracteres de uma string.

Log

Retorna o logaritmo natural de um número à base.

LTrim

Retorna uma cópia de uma string, com quaisquer espaços à sua

esquerda removidos.

Mid

Retorna uma string seguido um número especificado de caracteres.

Minute

Retorna a parte de minutos de um valor de tempo.

Month

Retorna o mês de um valor de data.

MsgBox

Exibe uma caixa de mensagem e (opcionalmente) retorna um valor.

Now

Retorna a data e horário do sistema atual.

RGB

Retorna um valor numérico RGB representando uma cor.

ExcelVBABokks.indb 134

15/02/2013 13:59:39

Capítulo 9: Usando VBA e Funções de Planilha 135

Tabela 9-2

(continuação)

Função

O Que Ela Faz

Replace

Substitui uma sub-string em uma string por outra sub-string.

Right

Retorna uma string conforme um número especificado de caracteres à sua direita.

Rnd

Retorna um número aleatório entre 0 e 1.

RTrim

Retorna uma cópia de uma string, com quaisquer espaços finais removidos.

Second

Retorna a parte de segundos de um valor de tempo.

Sgn

Retorna um número inteiro indicando o sinal do número.

Shell

Roda um programa executável.

Sin

Retorna o seno de um número.

Space

Retorna uma string com um número especificado de espaços.

Split

Divide em partes uma string, usando um caractere delimitador.

Sqr

Retorna a raiz quadrada de um número.

Str

Retorna a representação de um número de uma string.

StrComp

Retorna um valor indicando o resultado de uma comparação de string.

String

Retorna um caractere repetido ou string.

Tan

Retorna a tangente de um número.

Time

Retorna o horário atual do sistema.

Timer

Retorna o número de segundos desde a meia-noite.

TimeSerial Retorna o horário em hora, minuto ou segundo especificados.

TimeValue Converte uma string a um número serial de horário.

Trim

Retorna uma string sem espaços em seu início ou final.

TypeName Retorna uma string que descreve um tipo de dados variáveis.

UBound

Retorna o maior subscrito disponível à dimensão do array.

UCase

Converte uma string para maiúsculas.

Val

Retorna os números contidos em uma string.

VarType

Retorna um valor indicando o subtipo de uma variável.

Weekday Retorna um número representando um dia da semana.

Year

Retorna o ano a partir de um valor de data.

Usando Funções de Planilha no VBA

Ainda que o VBA ofereça uma boa diversidade de funções integradas, nem sempre você pode encontrar o que precisa. Felizmente, também é possível usar a maioria das funções de planilha do Excel em seus procedimentos VBA. As únicas funções de planilha que não podem ser usadas são aquelas que têm uma função VBA equivalente.

ExcelVBABokks.indb 135

15/02/2013 13:59:40

136 Parte III: Conceitos de Programação O VBA disponibiliza as funções de planilha do Excel através do objeto **BRE-SE**

WorksheetFunction, o qual está contido no objeto **Application**. Portanto,

to, qualquer declaração que use uma função de planilha deve usar o qualificador **Application.WorksheetFunction**. Em outras palavras, você deve preceder o nome da função com **Application.WorksheetFunction**, (com um ponto separando os dois).

A seguir está um exemplo: Total = Application.WorksheetFunction.Sum(Range("A1:A12"))

Você pode omitir a parte Application da parte WorksheetFunction da **DICA**

expressão. De qualquer modo, o VBA saberá o que você está fazendo.

Em outras palavras, as três expressões a seguir funcionam exatamente da mesma maneira:

Total = Application.WorksheetFunction.Sum(Range("A1:A12"))

Total = WorksheetFunction.Sum(Range("A1:A12"))

Total = Application.Sum(Range("A1:A12"))

A minha preferência pessoal é usar a parte WorksheetFunction apenas para deixar bem claro que o código está usando uma função Excel.

Exemplos de função e planilha

Nesta seção, demonstro como usar funções de planilha em suas expressões VBA.

Encontrando o valor máximo em uma faixa

Eis um exemplo que mostra como usar a função de planilha MÁXIMO do Excel em um procedimento VBA. Esse procedimento exibe o valor máximo encontrado na coluna A da planilha ativa:

```
Asub ShowMax()
```

```
Dim TheMax As Double
```

```
TheMax = WorksheetFunction.Max(Range("A:A"))
```

```
MsgBox
```

```
TheMax
```

```
End Sub
```

Você também pode usar a função MÍNIMO para obter o menor valor em uma faixa. E, como seria esperado, pode usar outras funções de planilha de maneira semelhante. Por exemplo, você pode usar a

função MAIOR para determinar o “n-ésimo” maior valor em uma faixa. A seguinte expressão demonstra isto: SecondHighest = WorksheetFunction.Large(Range(“A:A”),2)

Observe que a função LARGE usa dois argumentos; o segundo argumento representa o n-ésimo no caso, 2 (o segundo maior valor).

ExcelVBABokks.indb 136

15/02/2013 13:59:41

Capítulo 9: Usando VBA e Funções de Planilha 137

Calculando o pagamento de uma hipoteca

O próximo exemplo usa a função de planilha PGTO (na versão em inglês do excel: PMT) para calcular o pagamento de uma hipoteca. Eu uso três variáveis para armazenar os dados que são passados à função Pmt como argumentos. Uma caixa de mensagem exibe o pagamento calculado.

```
Sub PmtCalc()
```

```
Dim InRate As Double
```

```
Dim LoadAmt As Double
```

```
Dim Periods As Integer
```

```
InRate = 0.0825 / 12
```

```
Periods = 30 * 12
```

```
LoanAmt = 150000
```

```
MsgBox WorksheetFunction. _
```

```
Pmt(InRate,
```

```
Periods,
```

```
-LoanAmt)
```

```
End Sub
```

Como demonstra a declaração a seguir, você também pode inserir os

valores diretamente como argumentos de função:

```
MsgBox WorksheetFunction.Pmt(0.0825 / 12, 360, -150000)
```

No entanto, usar variáveis para armazenar os parâmetros torna o código mais fácil de ler e modificar, se necessário.

Usando uma função procv(lookup)

O exemplo a seguir usa funções InputBox e MsgBox do VBA, além da função PROCV (vlookup) do Excel. Ela solicita o número de um produto e depois obtém o preço através da consulta a uma tabela. Na Figura 9-1, a faixa A1:B13 é chamada de PriceList (lista de preço).

```
Sub GetPrice()
```

```
Dim PartNum As Variant
```

```
Dim Price As Double
```

```
PartNum = InputBox("Informe o número do produto")
```

```
Sheets("Prices").Activate
```

```
Price = WorksheetFunction. _
```

```
Vlookup(PartNum, Range("PriceList"), 2, False)
```

```
MsgBox PartNum & " costs " & Price
```

```
End Sub
```

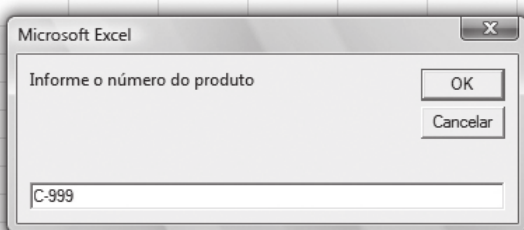
Você pode fazer o download desse livro de exercícios a partir do Web site do livro.

O procedimento começa assim:

1. A função InputBox do VBA pede o número do produto ao usuário.
2. Essa declaração atribui o número informado à variável PartNum.
3. A próxima declaração ativa a planilha Prices, só para o caso de ela ainda não ser uma planilha ativa.

	A	B	C	D	E	F	G	H	I	J
1	Produto	Preço								
2	A-132	39,95								
3	A-183	12,95								
4	B-942	16,49								
5	C-832	3,99								
6	C-999	17,59								
7	D-873	19,99								
8	F-143	39,95								
9	G-771	49,95								
10	K-873	129,95								
11	M-732	89,95								
12	P-101	3,95								
13	R-932	13,95								
14										
15										

Obtenha um preço



138 Parte III: Conceitos de Programação

4. O código usa a função VLOOKUP para encontrar o número do produto na tabela.

Observe que os argumentos que você usa nessa declaração são iguais àqueles que você usaria com a função em uma planilha de fórmula. Essa declaração atribui o resultado da função à variável Price.

5. O código exibe o preço do produto através da função MsgBox.

Figura 9-1:

A faixa

nomeada

como

PriceList

contém os

preços das

partes.

Este procedimento não tem nenhum erro de execução e ele falha

miseravelmente se você informar um número de produto inexistente (Tente). Se isso fosse um aplicativo, você poderia acrescentar algumas declarações de tratamento de erro para um procedimento mais robusto. No Capítulo 12, eu discuto sobre tratamento de erro.

Introduzindo funções de planilha

Você não pode usar a caixa de diálogo Inserir Função do Excel para inserir uma função de planilha em um módulo VBA. Ao invés disso, entre com tais funções à moda antiga: à mão. No entanto, você *pode* usar a caixa de diálogo Inserir Função para identificar a função que deseja usar e descobrir sobre seus argumentos.

Você também pode usufruir a vantagem da opção Autolistar membros
DICA

do VBE, a qual exibe uma lista drop-down de todas as funções de planilha. Basta digitar **Application.WorksheetFunction**, seguido por um ponto. Então, você verá uma lista das funções à sua disposição.

Se esse recurso não estiver funcionando, escolha o comando

Ferramentas⇒Opções do VBE, clique a guia Editor e marque a caixa de opções Autolistar Membros.

ExcelVBABokks.indb 138

15/02/2013 13:59:42

Capítulo 9: Usando VBA e Funções de Planilha 139

Mais Sobre o Uso de Funções de Planilha

Geralmente, os recém-chegados ao VBA confundem as funções integradas do VBA com as funções de planilha do Excel. Uma boa regra é lembrar que o VBA não tenta reinventar a roda. Na maior parte do tempo, o VBA não repete funções de planilha do Excel.

Para a maioria das funções de planilha que não estão disponíveis como métodos do objeto `WorksheetFunction`, você pode usar um operador ou função integrada a do VBA equivalente. Por exemplo, a função de planilha `Mod` não está disponível no objeto `WorksheetFunction`, pois o VBA tem um equivalente, o seu operador integrado `Mod`.

A questão toda? Se você precisa usar uma função, verifique primeiro se **DICA**

o VBA tem algo que atenda às suas necessidades. Se não, verifique as funções de planilha. Se tudo o mais falhar, você pode ser capaz de escrever uma função personalizada usando VBA.

Usando Funções Personalizadas

Eu abordei as funções VBA e as funções de planilha do Excel. A terceira categoria de funções que você pode usar em seus procedimentos VBA é a de funções personalizadas. Uma *função personalizada* (também conhecida como – Função Definida pelo Usuário) é aquela que você mesmo desenvolve usando (o que mais?) VBA. Para usar uma função personalizada, é preciso defini-la na pasta de trabalho na qual você a usa.

Eis um exemplo de definição de um simples procedimento Function e depois, o seu uso em um procedimento VBA Sub:

```
Function MultiplyTwo(num1, num2) As Double
```

```
MultiplyTwo = num1 * num2
```

```
End Function
```

```
Sub ShowResult()
```

```
Dim n1 As Double, n2 As Double
```

```
Dim Result As Double
```

```
n1 = 123
```

```
n2 = 544
```

```
Result = MultiplyTwo(n1, n2)
```

```
MsgBox Result
```

```
End Sub
```

A função personalizada MultiplyTwo tem dois argumentos. O procedimento Sub ShowResult usa esse procedimento Function, passando dois argumentos a ele (entre parênteses). Depois, o procedimento ShowResult exibe uma caixa de mensagem, mostrando o valor retornado pela função MultiplyTwo.

15/02/2013 13:59:43

140 Parte III: Conceitos de Programação Provavelmente, eu não preciso dizer a você que a função `MultiplyTwo` é bem inútil. É muito mais eficaz executar a multiplicação no procedimento `Sub ShowResult`. Eu a incluo para que você tenha uma ideia de como um procedimento `Sub` pode usar uma função personalizada.

Você também pode usar funções personalizadas em suas fórmulas de planilha. Por exemplo, se `MultiplyTwo` estiver definida em sua pasta de trabalho, você pode escrever uma fórmula, tal como esta:

```
=MultiplyTwo(A1, A2)
```

Esta fórmula retorna o produto dos valores nas células A1 e A2.

As funções de planilha personalizadas são um tópico importante (e bem útil). Tão importante (e útil) que dedico um capítulo inteiro a ele.

Vejam o Capítulo 20.

ExcelVBABokks.indb 140

15/02/2013 13:59:43

Capítulo 10

Controlando o Fluxo

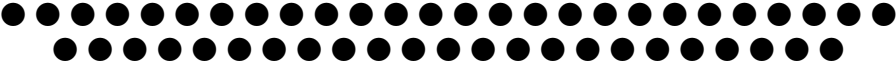
de Programa e

Tomando Decisões



Neste Capítulo

- ☐ Como descobrir métodos para controlar o fluxo de suas rotinas VBA
- ☐ Como saber sobre a temida declaração `GoTo`
- ☐ Usando as estruturas `If-Then` e `Select Case`
- ☐ Como executar loop em seus procedimentos



A lguns procedimentos VBA começam no início do código e progri-
dem, linha por linha, até o final, nunca se desviando desse fluxo de
programa, de cima para baixo. As macros que você grava sempre
funcionam assim. Porém, em muitos casos, é preciso controlar o fluxo
do seu código, pulando algumas declarações, executando algumas
declarações diversas vezes e testando condições para determinar qual
procedimento executar em seguida. Segure-se e aproveite o passeio,
pois você está prestes a descobrir a essência da programação.

Seguindo o Fluxo, Cara

Alguns novatos em programação não podem entender como um
computador burro pode tomar decisões inteligentes. O segredo está
em várias montagens de programação que a maioria das linguagens de
programação suporta. A Tabela 10-1 oferece um resumo dessas constru-

ções. Mais adiante neste capítulo, eu explico todas elas.

ExcelVBABokks.indb 141

15/02/2013 13:59:43

142 Parte III: Conceitos de Programação Tabela 10-1 Construções de Programação para tomar Decisões

Construção

Como Funciona

Declaração GoTo Pula para uma declaração em especial.

Estrutura If-Then Faz alguma coisa se algo mais for verdadeiro.

Select Case

Multiuso, mas depende do valor atribuído..

Loop For-Next

Executa uma série de declarações um número de vezes
especificado.

Loop DoWhile

Faz algo, desde que alguma outra coisa continue a ser verdadeira.

Loop DoUntil

Faz algo até que alguma outra coisa se torne verdadeira.

A Declaração GoTo

A declaração GoTo oferece o meio mais direto de alterar o fluxo de um programa. A declaração GoTo simplesmente transfere a execução do programa para uma nova declaração, a qual é precedida por uma etiqueta.

As suas rotinas VBA podem conter tantas etiquetas quantas você quiser. Uma *label* (etiqueta) é apenas uma string de texto seguida por dois pontos.

O seguinte procedimento mostra como funciona uma declaração GoTo: Sub GoToDemo()

```
UserName = InputBox("Insira seu nome: ")
```

```
If UserName <> "Bill Gates" Then GoTo WrongName
```

```
MsgBox ("Bem-vindo Bil...")
```

```
' ...[Mais código aqui] ...
```

```
Exit Sub
```

```
WrongName:
```

```
MsgBox "Desculpe.Somente Bill Gates pode rodar  
este programa."
```

```
End Sub
```

O procedimento usa a função InputBox para obter o nome do usuário.

Se o usuário entrar com um nome diferente de Bill Gates, o fluxo do programa pula para a etiqueta WrongName (nome errado), exibe uma mensagem de desculpas e o procedimento termina. Por outro lado, se

o Sr. Gates rodar esse procedimento e usar o seu verdadeiro nome, o procedimento exibe uma mensagem de boas vindas e depois executa algum código adicional (não mostrado no exemplo). Observe que a declaração Exit Sub encerra o procedimento antes que a segunda função MsgBox tenha uma chance de trabalhar.

ExcelVBABokks.indb 142

15/02/2013 13:59:44

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 143

DE ESPECIALISTA

O P A P

O que é programação estruturada?

Isso importa?

Se você anda com programadores, cedo ou fácil de acompanhar – diferente de um pro-tarde ouvirá o termo *programação estrutu-*

grama que pula em volta de modo desorde-

rada. Este termo existe há décadas e em nado. Praticamente isso elimina o uso da geral, os programadores concordam que declaração GoTo.

programas estruturados são superiores a

programas não estruturados. Assim, o que é Geralmente, um programa estruturado é programação estruturada? E é possível fa-mais fácil de ler e de entender. Mais impor—

zê-la usando VBA?

tante, ele também é mais fácil de modificar

quando surge a necessidade.

A premissa básica de programação estruturada é que uma rotina ou segmento de códi—

De fato, VBA é uma linguagem estruturada.

go só deve ter um ponto de entrada e um de saída. Ela oferece montagens padrão estrutura-saída. Em outras palavras, um bloco de có-

das, tais como as estruturas de loops Then—

digo deve ser uma unidade individual. Um -Else, For-Next, DoUntil, DoWhile e Select programa não pode pular no meio dessa Case. Além do mais, ela suporta totalmente unidade, nem pode sair em qualquer ponto, construções de código de módulos. Se você exceto no único ponto de saída. Quando for novo em programação, deveria tentar você escreve código estruturado, o seu pro-desenvolver bons hábitos de programação grama progride de uma forma ordenada e é estruturada logo no início. Fim da aula.

Esta simples rotina funciona, mas o VBA oferece diversas alternativas melhores (e mais estruturadas) do que GoTo. Em geral, você só deveria usar GoTo quando não tiver outra forma de executar uma ação. Na vida real, a única vez em que você deve usar uma declaração GoTo é para driblar erros (eu discuto isso no Capítulo 12).

Muitos tipos de programação explícitos têm uma aversão intrínseca por **CUIDADO!**

declarações GoTo, porque usá-las pode resultar em “código espagueti”

difícil de ler (e difícil de manter). Portanto, você deveria evitar esse assunto ao falar com outros programadores.

Decisões, decisões

Nesta seção, discuto duas estruturas de programação que podem reforçar os seus procedimentos VBA quanto à capacidade de tomar decisões: If-Then e Select Case.

A estrutura If-Then

Certo, eu direi: If-Then é a estrutura de controle mais importante de VBA. Provavelmente, você usa esse comando diariamente (pelo menos, *eu* uso). Como muitos outros aspectos da vida, tomar decisões eficientes é a chave do sucesso em escrever macros do Excel. Se este livro tiver o efeito que pretendo, logo você compartilhará a minha filosofia ExcelVBABokks.indb 143

144 Parte III: Conceitos de Programação de que um aplicativo Excel bem sucedido concentra-se em tomar decisões e agir de acordo com elas.

A estrutura If-Then tem esta sintaxe básica:

If condition Then statements (Else elsestatements)

Use a estrutura If-Then quando quiser executar, condicionalmente, uma ou mais declarações. A cláusula opcional Else, se incluída, permite que você execute uma ou mais declarações se a condição que estiver testando não for verdadeira. Parece confuso? Não se preocupe: alguns exemplos esclarecerão isso.

Exemplos de If-Then

A rotina a seguir demonstra a estrutura If-Then sem a cláusula opcional Else: Sub GreetMe()

```
If Time < 0.5 Then MsgBox "Bom dia"
```

```
End Sub
```

O procedimento GreetMe (cumprimente-me) usa a função Time do VBA para obter o horário do sistema. Se o horário atual do sistema for menor que .5 (em outras palavras, antes do meio-dia), a rotina exibe uma saudação amigável. Se Time for maior ou igual a .5, a rotina termina e nada acontece.

Para exibir uma saudação diferente, se Time for maior que ou igual a .5, acrescente uma outra declaração If-Then após a primeira:

```
Sub GreetMe2()
```

```
If Time < 0.5 Then MsgBox "Bom dia"
```

```
If Time >= 0.5 Then MsgBox "Boa tarde"
```

```
End Sub
```

Observe que eu usei $\geq$ (maior ou igual a) para a segunda declaração If-Then. Isso garante que o dia inteiro está coberto. Se eu tivesse usado $>$ (maior que), então nenhuma mensagem apareceria se esse procedimento fosse executado exatamente às 12 horas. Isso é bem improvável, mas com um programa importante como esse, não quero correr quaisquer riscos.

Um exemplo de If-Then-Else

Uma outra abordagem ao problema anterior usa a cláusula Else. Eis a mesma rotina, recodificada para usar a estrutura If-Then-Else:

```
Sub GreetMe3()
```

```
If Time < 0.5 Then MsgBox “Bom dia” Else _
```

```
MsgBox “Boa tarde”
```

```
End Sub
```

ExcelVBABokks.indb 144

15/02/2013 13:59:44

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 145

Veja que eu uso o caractere de continuação de linha (sublinhado) no exemplo anterior. Na verdade, a declaração If-Then-Else é uma única declaração. VBA oferece uma maneira ligeiramente diferente de codificar montagens If-Then-Else que usam uma declaração End-If (terminar se). Portanto, o procedimento GreetMe pode ser reescrito como: Sub GreetMe4() If Time < 0.5 Then

```
MsgBox “Bom dia”
```

```
Else
```

```
MsgBox “Boa tarde”
```

```
End If
```

```
End Sub
```

Na verdade, é possível inserir qualquer quantidade de declarações depois do If, e qualquer quantidade de declarações depois do Else. Eu

prefiro usar essa sintaxe, pois ela é mais fácil de ler e torna as declarações mais curtas.

E se você precisar expandir a rotina GreetMe para lidar com três condi-

ções: manhã, tarde e noite? Você tem duas opções: usar três declarações If-Then ou usar uma estrutura *aninhada* de If-Then-Else. *Aninhar* significa colocar uma estrutura If-Then-Else dentro de outra estrutura If-Then-Else.

Na primeira abordagem, é mais simples usar as três declarações: Sub GreetMe5()

```
Dim Msg As String
```

```
If Time < 0.5 Then Msg = "Manhã"
```

```
If Time >= 0.5 And Time < 0.75 Then Msg = "Tarde"
```

```
If Time >= 0.75 Then Msg = "Noite"
```

```
MsgBox "Boa " & Msg
```

```
End Sub
```

A variável Msg obtém um texto de valor diferente, dependendo da hora do dia. A declaração MsgBox final exibe a saudação Good Morning (bom dia), Good Afternoon (boa tarde) ou Good Evening (boa noite).

A seguinte rotina executa a mesma ação, mas usa a estrutura If-Then-End If:

```
Sub GreetMe6()
```

```
Dim Msg As String
```

```
If Time < 0.5 Then
```

```
Msg = "Manhã"
```

```
End If
```

```
If Time >= 0.5 And Time < 0.75 Then
```


Msg = “Tarde”

End If

If Time >= 0.75 Then

Msg = “Noite”

End If

MsgBox “Boa “ & Msg

End Sub

ExcelVBABokks.indb 145

15/02/2013 13:59:44

146 Parte III: Conceitos de Programação Quão rápidos são os loops?

Você poderia estar curioso sobre quão rápido o VBA pode rodar através de loops com If-

-Then. Alguns sistemas rodam código significativamente mais depressa do que outros?

Como uma experiência informal, eu postei o seguinte procedimento VBA em meu blog e pedi aos outros para postar seus resultados:

Sub TimeTest()

‘100 milhões de números aleatórios, testes e operações

matemáticas

Dim x As Long

Dim StartTime As Single

Dim i As Long

x = 0

StartTime = Timer

For i = 1 To 100000000

```
If Rnd <= 0.5 Then x = x + 1 Else X = x - 1
```

```
Next
```

```
i
```

```
MsgBox Timer - StartTime & " seconds"
```

```
End Sub
```

O código faz loops 100 milhões de vezes e executa algumas operações dentro do loop: ele gera um número aleatório, faz uma comparação If-Then e realiza uma operação matemática. Quando o loop é concluído, o tempo transcorrido é exibido em uma caixa de mensagem.

O meu sistema rodou através desse loop em 9.03 segundos. Cerca de 100 outras pessoas postaram seus resultados e os tempos variaram de aproximadamente 5 a 30 segundos. Em outras palavras, alguns computadores são aproximadamente seis vezes mais rápidos do que outros. É bom saber isso.

Porém, a verdadeira dúvida é, quanto tempo eu levaria para fazer isso manualmente? Escrevi — em um pedaço de papel e joguei uma moeda. Se desse cara, eu acrescentaria um à minha conta. Se desse coroa, eu subtrairia 1. Eu fiz isso dez vezes e demorei 42 segundos.

Portanto, através do meu “loop” demorou 4.2 segundos. Usando essa informação, calculei que eu demoraria 799 anos para executar essa tarefa 100 milhões de vezes – mas só se eu trabalhasse sem parar. A conclusão: meu computador é aproximadamente 52.3 milhões de vezes mais rápido do que eu.

Usando ElseIf

Nos exemplos anteriores, cada declaração na rotina é executada —

mesmo de manhã. Uma estrutura mais eficiente sairia da rotina assim que descobrisse uma condição como verdadeira. Por exemplo, pela manhã, o procedimento exibiria a mensagem Good Morning e depois sairia — sem avaliar as outras condições supérfluas.

Com uma pequena rotina como essa, você não precisa se preocupar com a velocidade de execução. Mas, em aplicativos maiores, onde a velocidade é importante, você deveria saber sobre uma outra sintaxe para a estrutura If-Then. A sintaxe de ElseIf é: ExcelVBABokks.indb

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 147

If condition Then

[*statements*]

ElseIf condition-n Then

[elseifstatements]

Else

[elsestatements]

End If

Eis como você pode reescrever a rotina GreetMe usando esta sintaxe:
Sub GreetMr7()

Dim Msg As String

If Time < 0.5 Then

Msg = “Manhã”

ElseIf Time >= 0.5 And Time < 0.75 Then

Msg = “Tarde”

Else

Msg = “Noite”

End If

MsgBox “Boa “ & Msg

End Sub

Quando uma condição é verdadeira, o VBA executa as declarações condicionais e a estrutura If termina. Em outras palavras, o VBA não perde tempo avaliando as condições estranhas, o que torna esse procedimento um pouco mais eficiente que os exemplos anteriores. A

divergência (divergências sempre existem) é que o código é mais difícil de entender (claro que você já sabia disso).

Uma pasta de trabalho que contém tudo sobre os exemplos GreetMe pode ser baixada do site deste livro.

Um outro exemplo If-Then

Eis um outro exemplo que usa a forma simples da estrutura If-Then.

Esse procedimento solicita ao usuário uma quantidade e, depois, exibe o desconto apropriado com base na quantidade fornecida pelo usuário.

Sub ShowDiscount()

Dim Quantity As Integer

Dim Discount As Double

Quantity = InputBox("Digite a quantidade:")

If Quantity > 0 Then Discount = 0.1

If Quantity >= 25 Then Discount = 0.15

If Quantity >= 50 Then Discount = 0.2

If Quantity >= 75 Then Discount = 0.25

MsgBox "Desconto: " & Discount

End Sub

Uma pasta de trabalho contendo os exemplos desta seção pode ser baixada a partir do Web site deste livro.

Observe que cada declaração If-Then nesta rotina é executada e que o valor de Discount pode mudar, enquanto as declarações são executadas.

ExcelVBABokks.indb 147

15/02/2013 13:59:44

148 Parte III: Conceitos de Programação das. No entanto, a rotina exibe o valor correto de Discount, pois eu coloquei as declarações If-Then em ordem crescente de valores.

O procedimento a seguir executa as mesmas tarefas, usando a sintaxe

Else alternativa. Neste caso, a rotina termina imediatamente depois de executar as declarações quando a condição é verdadeira.

```
Sub ShowDiscount2()
```

```
Dim Quantity As Integer
```

```
Dim Discount As Double
```

```
Quantity = InputBox("Digite a quantidade: ")
```

```
If Quantity > 0 And Quantity < 25 Then
```

```
Discount = 0.1
```

```
ElseIf Quantity >= 25 And Quantity < 50 Then
```

```
Discount = 0.15
```

```
ElseIf Quantity >= 50 And Quantity < 75 Then
```

```
Discount = 0.2
```

```
ElseIf Quantity >= 75 Then
```

```
Discount = 0.25
```

```
End If
```

```
MsgBox "Desconto: " & Discount
```

```
End Sub
```

Pessoalmente, eu acho essas múltiplas estruturas If-Then bem ineficientes. Em geral, uso a estrutura If-Then apenas para decisões de binário simples. Quando uma decisão envolve três ou mais escolhas, a estrutura Select Case oferece uma abordagem mais simples, mais eficiente.

A estrutura Select Case

A estrutura Select Case é útil em decisões envolvendo três ou mais opções (embora ela só funcione com duas opções, oferecendo uma alternativa à estrutura If-Then-Else).

A seguir, a sintaxe para a estrutura Select Case:

```
Select Case testexpression
```

[Case expressionlist-n

[statements-n]] ...

[Case Else

[elsestatements]]

End Select

Não fique assustado por esta sintaxe oficial. Usar a estrutura Select Case é bem fácil.

Um exemplo de Select Case

O exemplo a seguir mostra como usar a estrutura Select Case. Isso também mostra uma outra forma de codificar os exemplos apresentados na seção anterior.

ExcelVBABokks.indb 148

15/02/2013 13:59:44

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 149

Sub ShowDiscount3()

Dim Quantity As Integer

Dim Discount As Double

Quantity = InputBox("Digite a quantidade: ")

Select Case Quantity

Case 0 To 24

Discount

=

Case 25 To 49

Discount

=

0.15

Case 50 To 74

Discount

=

0.2

Case Is > = 75

Discount

=

0.25

End Select

MsgBox “Desconto: “ & Discount

End Sub

Neste exemplo, a variável Quantity (quantidade) está sob avaliação. A rotina está verificando os quatro casos diferentes (0-24, 25-49, 50-74 e 75 ou maior).

Qualquer quantidade de declarações pode seguir cada declaração Case, e todas elas são executadas se a condição for verdadeira. Se você usar apenas uma declaração, como neste exemplo, pode colocar a declara-

ção na mesma linha que a palavra chave Case, precedida por dois pontos — um caractere separador de declaração de VBA. Na minha opinião, isso torna o código mais compacto e um pouco mais claro. Eis

como se parece a rotina, usando este formato: Sub ShowDiscount4()

Dim Quantity As Integer

Dim Discount As Double

Quantity = InputBox("Digite a quantidade: ")

Select Case Quantity

Case 0 To 24: Discount = 0.1

Case 25 To 49: Discount = 0.15

Case 50 To 74: Discount = 0.2

Case Is >= 75: Discount = 0.25

End Select

MsgBox "Desconto: " & Discount

End Sub

Quando o VBA executa uma estrutura Select Case, a estrutura é finalizada assim que o VBA encontra um caso verdadeiro e executa as declara-

ções para aquele caso.

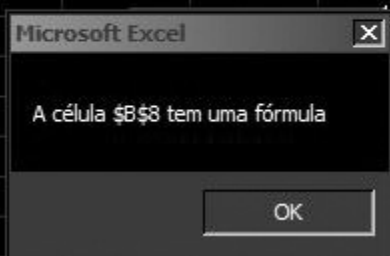
Um exemplo de Select Case aninhada

Conforme demonstrado no exemplo a seguir, é possível aninhar estruturas Select Case. Essa rotina examina a célula ativa e exibe uma mensagem descrevendo o conteúdo da célula. Observe que o procedimento tem três estruturas Select Case e cada uma tem sua própria declaração End Select.

ExcelVBABokks.indb 149

15/02/2013 13:59:45

	A	B	C	D	E	F
1						
2	Jan	32				
3	Feb	0				
4	Mar	5				
5	Apr	43				
6	May	83				
7	Jun	122				
8	Total	285				
9						



150 Parte III: Conceitos de Programação

Sub CheckCell()

Dim Msg As String

Select Case IsEmpty(ActiveCell)

Case

True

Msg

=

“está

vazia.”

Case

Else

Select

Case

ActiveCell.HasFormula

Case

True

Msg

=

“tem

uma

fórmula”

Case

False

Select

Case

IsNumeric(ActiveCell)

Case

True

Msg

=

“tem

um

número”

Case

Else

Msg

=

“tem

texto”

End

Select

End

Select

End

Select

MsgBox “célula “ & ActiveCell.Address& “ “ & Msg End Sub

Este exemplo está disponível no Web site deste livro.

A lógica é mais ou menos assim:

1. Descobrir se a célula está vazia.
2. Se não estiver vazia, veja se ela contém uma fórmula.
3. Se não houver fórmula, descobrir se ela contém um valor numérico ou texto.

Quando a rotina termina, a variável Msg contém uma string que descreve o conteúdo da célula. Conforme mostrado na Figura 10-1, a função MsgBox exibe aquela mensagem.

Você pode aninhar estruturas Select Case tão profundamente quanto precisar, mas, assegure-se de que cada declaração Select Case tenha uma declaração End Select correspondente.

Figura 10-1:

Uma

mensagem

exibida pelo

procedimento

CheckCell

ExcelVBABokks.indb 150

15/02/2013 13:59:45

Decisões 151

Se você tiver pensado sobre a razão dos recuos no código que apresentei

aqui, a listagem anterior serve como um bom exemplo. Na verdade, os recuos ajudam a esclarecer os níveis de aninhamento (pelo menos, eu acho). Se você não acredita em mim, dê uma olhada no mesmo procedimento, sem qualquer recuo: Sub CheckCell()

```
Dim Msg As String
```

```
Select Case IsEmpty(ActiveCell)
```

```
Case True
```

```
Msg = “está vazia.”
```

```
Case Else
```

```
Select Case ActiveCell.HasFormula
```

```
Case True
```

```
Msg = “tem uma fórmula”
```

```
Case False
```

```
Select Case IsNumeric(ActiveCell)
```

```
Case True
```

```
Msg = “tem um número”
```

```
Case Else
```

```
Msg = “tem texto”
```

```
End Select
```

```
End Select
```

```
End Select
```

```
MsgBox “Célula “ & ActiveCell.Address & “ “ & Msg End Sub
```

Bem incompreensível, não é?

O termo *looping* refere-se a repetir, várias vezes, um bloco de declarações VBA. Nesta seção, explico sobre os vários tipos diferentes de loops.

Há dois tipos de loops: loops bons e loops ruins (os loops bons são recompensados e os ruins são mandados para os seus quartos).

O código a seguir demonstra um loop ruim. O procedimento apenas entra com números consecutivos em uma faixa. Ele inicia, solicitando dois valores ao usuário um valor de partida e o número total de células a preencher (pelo fato de `InputBox` retornar uma string, eu converto as strings em inteiros, usando a função `Cint`). Essa loop usa a declaração `GoTo` para controlar o fluxo. A variável `CellCount` controla quantas células são preenchidas. Se esse valor for menor que o número informado pelo usuário, o controle do programa faz um loop de volta para `DoAnother`.

ExcelVBABokks.indb 151

15/02/2013 13:59:45

152 Parte III: Conceitos de Programação Sub BadLoop()

```
Dim StartVal As Long
```

```
Dim NumToFill As Long
```

```
Dim CellCount As Long
```

```
StartVal = InputBox("Insira o valor inicial: ")
```

```
NumToFill = InputBox("Quantas células? ")
```

```
ActiveCell = StartVal
```

```
CellCount = 1
```

```
DoAnother:
```

```
ActiveCell.Offset(CellCount, 0) = StartVal +
```

```
CellCount
```

```
CellCount = CellCount + 1
```

If CellCount < NumToFill Then GoTo DoAnother _

Else Exit Sub

End Sub

Esta rotina funciona conforme planejado, mas eu não estou particularmente orgulhoso dela. Então, por que esse é um exemplo de loop ruim?

Conforme mencionei anteriormente neste capítulo, evite usar uma declaração GoTo, a menos que seja absolutamente necessário. Usar declarações para executar loop

5 É contrário ao conceito de programação estruturada (veja o artigo apresentado anteriormente neste capítulo “O que é programação estruturada? Isso importa?”)

5 Torna o código mais difícil de ler.

5 É mais propenso a erros do que usando procedimentos estruturados de loop.

VBA tem tantos comandos estruturados de loop, que quase nunca você precisa se basear em declarações GoTo para tomar suas decisões. De novo, a exceção está no tratamento de erros.

Agora, você pode passar para uma discussão de estruturas de loop boa.

Loop For-Next

O tipo mais simples de loop é o For-Next. Eis a sintaxe para essa estrutura:

```
For counter = start To end [ Step stepval]  
[ statements]
```

[Exit For]

[*statements*]

Next [*counter*]

,

O loop é controlado por uma variável counter (contador), a qual começa em um valor e acaba em outro. As declarações entre For e Next ExcelVBABokks.indb 152

15/02/2013 13:59:45

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 153

são declarações que se repetem no loop. Para ver isso funcionar, continue lendo.

Um exemplo For-Next

O seguinte exemplo mostra um loop For-Next que não usa o valor opcional Step ou a declaração opcional Exit. Essa rotina faz loops 20

vezes e usa a função Rnd para entrar com um número aleatório em 20 células, começando com a célula ativa:

```
Sub FillRange()
```

```
Dim Count As Long
```

```
For Count = 0 To 19
```

```
ActiveCell.Offset(Count, 0) = Rnd
```

```
Next Count
```

```
End Sub
```

Neste exemplo, Count (a variável que conta os loops) é iniciada com o valor 0 e aumenta em 1 cada vez que passa pela loop. Pelo fato de que eu não especifiquei um valor Step, VBA usa o valor (1) padrão. O

método Offset usa o valor de Count como um argumento. Na primeira vez que passa pelo loop, Count é 0 e o procedimento entra com um número offset (a diferença entre um valor atual e um valor desejado) na célula ativa em zero linhas. Na segunda vez que passa (Count = 1), o procedimento entra com um número em offset na célula ativa em uma linha, e assim por diante.

Porque o contador de loop é uma variável normal, você pode escrever código para alterar o seu valor dentro do bloco de códigos entre as **CUIDADO!**

declarações For e Next. No entanto, essa é uma prática *muito ruim*.

Alterar o contador dentro do loop pode levar a resultados imprevisíveis. Tenha cuidado especial para garantir que o seu código não altere diretamente o valor do contador de loop.

Um exemplo de For-Next com um Step

Você pode usar um valor Step para pular alguns valores em um loop For-Next. Eis o mesmo procedimento da seção anterior, reescrito para inserir números aleatórios em células intercaladas: Sub FillRange2()

```
Dim Count As Long
```

```
For Count = 0 To 19 Step 2
```

```
ActiveCell.Offset(Count, 0) = Rnd
```

```
Next Count
```

```
End Sub
```

Dessa vez, Count começa como 0 e depois toma o valor 2, 4, 6 e assim por diante. O valor final de Count é 18. O valor Step determina como o contador é incrementado. Veja que o valor superior do loop (19) não é usado, pois o valor mais alto de Count depois de 18 seria 20, e 20 é maior do que 19.

A Figura 10-2 mostra o resultado obtido ao executar FillRange2 quando a célula B2 é a célula ativa.

ExcelVBABokks.indb 153

15/02/2013 13:59:45

	A	B	C	D	E
1					
2		0.045353			
3					
4		0.414033			
5					
6		0.862619			
7					
8		0.79048			
9					
10		0.373536			
11					
12		0.961953			
13					
14		0.871446			
15					
16		0.056237			
17					
18		0.949557			
19					
20		0.364019			
21					
22					
23					

154 Parte III: Conceitos de Programação

Anteriormente neste capítulo, você viu o exemplo BadLoop (loop ruim), o qual usa uma declaração GoTo. Eis o mesmo exemplo, que está disponível no site deste livro, convertido a um loop bom, usando a estrutura For-Next: Sub GoodLoop()

```
Dim StartVal As Long
```

```
Dim NumToFill As Long
```

```
Dim CellCount As Long
```

```
StartVal = InputBox("Insira o valor inicial: ")
```

```
NumToFill = InputBox("Quantas células? ")
```

```
For CellCount = 1 To NumToFill
```

```
ActiveCell.Offset(CellCount - 1, 0) = _
```

```
StartVal + CellCount - 1
```

```
Next CellCount
```

```
End Sub
```

Figura 10-2:

Usando um

loop para

gerar

números

aleatórios.

Um exemplo For-Next com uma declaração Exit For

Um loop For-Next também pode incluir uma ou mais declarações Exit For dentro do loop. Quando VBA encontra essa declaração, o loop termina imediatamente.

O exemplo seguinte, disponível no site do livro, demonstra a declaração Exit For. Essa rotina identifica quais células da planilha ativa na coluna A têm o maior valor:

```
Sub ExitForDemo()
```

```
Dim MaxVal As Double
```

```
Dim Row As Long
```

```
MaxVal = WorksheetFunction.Max(Range("A:A"))
```

```
For Row = 1 To Rows.Count
```

```
ExcelVBABokks.indb 154
```

```
15/02/2013 13:59:45
```

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 155

```
If Range("A1").Offset(Row-1, 0).Value = MaxVal Then
```

```
Range("A1").Offset(Row-1,
```

```
0).Activate
```

```
MsgBox "Max value is in Row " & Row
```

```
Exit
```

```
For
```

```
End
```

```
If
```

```
Next Row
```

```
End Sub
```

Essa rotina calcula o valor máximo na coluna, usando a função MAX do Excel e atribui o resultado à variável MaxVal (valor máximo). Depois, a loop For-Next verifica cada célula na coluna. Se a célula sendo verificada for igual a MaxVal, a rotina não precisa continuar a fazer loop (o seu trabalho está encerrado), assim a declaração Exit For encerra a loop.

Agora, você poderia gritar, “Ei, mas você disse algo sobre usar sempre um único ponto de saída!”. Bem, você está certo e, obviamente, está começando a entender esse negócio de programação estruturada. Mas, em alguns casos, é uma decisão sábia ignorar essa regra. Neste exemplo, isso aumentará muito a velocidade de seu código, porque não há motivo para continuar o loop depois do valor ser encontrado.

Antes de encerrar o loop, o procedimento ativa a célula com o valor máximo e informa ao usuário a localização dele.

Observe que eu uso Rows.Count (contagem de linhas) na declaração For. A propriedade de contar do objeto Rows retorna o número de linhas na planilha. Portanto, você pode usar esse procedimento com versões anteriores de Excel (que têm menos linhas).

Um exemplo For-Next aninhado

Até agora, todos os exemplos deste capítulo usam loops relativamente simples. Porém, você pode ter qualquer quantidade de declarações em um loop e aninhar loops For-Next dentro de outros loops For-Next.

O seguinte exemplo usa um loop aninhado For Next para inserir números aleatórios em uma faixa de células de 12-linhas por 5 colunas, conforme mostrado na Figura 10-3. Observe que a rotina executa o *loop interno* (o loop com o contador Row), para cada iteração do *loop externo* (o loop com o contador Col). Em outras palavras, a rotina executa 60 vezes a declaração Cells(Row, Col) = Rnd.

```
Sub FillRange2()
```

```
Dim Col As Long
```

```
Dim Row As Long
```

```
For Col = 1 To 5
```

```
For Row = 1 To 12
```

```
Cells(Row, Col) = Rnd
```

```
Next
```

```
Row
```

```
Next Col
```

```
End Sub
```

ExcelVBABokks.indb 155

15/02/2013 13:59:45

	A	B	C	D	E	F	G	H
1	0,705548	0,862619	0,4687	0,695116	0,28448			
2	0,533424	0,79048	0,298165	0,980003	0,045649			
3	0,579519	0,373536	0,622697	0,243931	0,295773	Preenche o intervalo		
4	0,289562	0,961953	0,647821	0,533873	0,382011			
5	0,301948	0,871446	0,263793	0,10637	0,30097			
6	0,77474	0,056237	0,279342	0,999415	0,948571			
7	0,014018	0,949557	0,829802	0,676176	0,979829			
8	0,760724	0,364019	0,824602	0,015704	0,401374			
9	0,81449	0,524868	0,589163	0,575184	0,27828			
10	0,709038	0,767112	0,986093	0,100052	0,160442			
11	0,045353	0,053505	0,910964	0,103023	0,162822			
12	0,414033	0,592458	0,226866	0,798884	0,646587			

156 Parte III: Conceitos de Programação

Figura 10-3:

Estas

células

foram

preenchidas usando

um loop

For-Next

aninhado.

O próximo exemplo usa loops For-Next aninhados para inicializar um array tridimensional com o valor 100. Essa rotina executa a declaração no meio de todas as loops (a declaração de atribuição) 1.000 vezes ($10 \times 10 \times 10$), cada vez com uma combinação de valores diferentes para i, j e k: Sub NestedLoops() Dim MyArray(10, 10, 10)

Dim i As Integer

Dim j As Integer

Dim k As Integer

For i = 1 To 10

For j = 1 To 10

For k = 1 To 10

MyArray(i, j, k) = 100

Nest

k

Next

j

Next i

‘ Other statements go here

End Sub

Para informações sobre arrays, volte ao Capítulo 7.

Loop DoWhile

VBA suporta um outro tipo de estrutura de loop, conhecida como uma loop DoWhile. Diferente de uma loop For-Next, uma loop DoWhile continua até que seja atingida uma condição especificada. Eis a sintaxe da loop DoWhile: Do [While *condition*]

[*statements*]

[Exit Do]

[*statements*]

Loop

ExcelVBABokks.indb 156

15/02/2013 13:59:45

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 157

O exemplo a seguir usa um loop DoWhile. Esta rotina usa a célula ativa como ponto de partida e então segue coluna abaixo, multiplicando cada célula por 2. O loop continua até que a rotina encontre uma célula vazia.

```
Sub DoWhileDemo()
```

```
Do While ActiveCell.Value <> Empty
```

```
ActiveCell.Value = ActiveCell.Value * 2
```

```
ActiveCell.Offset(1, 0).Select
```

```
Loop
```

```
End Sub
```

Algumas pessoas preferem codificar um loop DoWhile como um loop Do-Loop While. Este exemplo age exatamente da mesma forma que o procedimento anterior, mas usa uma sintaxe de loop diferente: Sub DoLoopWhileDemo() Do

```
ActiveCell.Value = ActiveCell.Value * 2
```

```
ActiveCell.Offset(1, 0).Select
```

```
Loop While ActiveCell.Value <> Empty
```

```
End Sub
```

A principal diferença entre os loops DoWhile e Do-Loop While: o loop **BRE-SE**

DoWhile sempre roda o seu teste condicional primeiro. Se o teste não **LEM**

for verdadeiro, as instruções dentro do loop nunca são executadas. O

loop Do-Loop While, em contrapartida, sempre executa o seu teste condicional depois que as instruções anteriores ao loop forem executadas. Portanto, as instruções do loop sempre são executadas pelo menos uma vez, independente do teste. A diferença pode, às vezes, ter um grande efeito sobre o funcionamento do seu programa.

Loop DoUntil

A estrutura do loop DoUntil é similar a estrutura do loop DoWhile. As duas estruturas diferem quanto a verificação das condições testadas.

Um programa continua executando um loop DoWhile, enquanto a condição permanecer verdadeira. Em um loop DoUntil, o programa executa o loop até que a condição seja verdadeira.

Aqui está a sintaxe DoUntil:

Do [*While condition*]

[*statements*]

[Exit Do]

[*statements*]

Loop

O seguinte exemplo é o mesmo apresentado para a loop DoWhile, porém, recodificado para usar a loop DoUntil:

ExcelVBABokks.indb 157

15/02/2013 13:59:46

158 Parte III: Conceitos de Programação Sub DoUntilDemo()

Do Until IsEmpty(ActiveCell.Value)

ActiveCell.Value = ActiveCell.Value * 2

ActiveCell.Offset(1, 0).Select

Loop

End Sub

Exatamente como com o loop DoWhile, você pode encontrar uma forma diferente do loop DoUntil. O seguinte exemplo, o qual tem o mesmo efeito que o procedimento anterior, demonstra uma sintaxe alternativa para esse tipo de loop:

```
Sub DoLoopUntilDemo()
```

```
Do
```

```
ActiveCell.Value = ActiveCell.Value * 2
```

```
ActiveCell.Offset(1, 0).Select
```

```
Loop Until IsEmpty(ActivateCell.Value)
```

```
End Sub
```

Existe uma diferença sutil em como o loop DoUntil e o Do-Loop Until operam. Na primeira, o teste é executado no início do loop, antes de **CUIDADO!**

qualquer coisa no corpo do loop. Isso significa que é possível que o código no corpo do loop não ser executado se a condição de teste for atingida. Na última versão, a condição é testada ao final do loop.

Portanto, no mínimo, o loop Do-Loop sempre resulta no corpo do loop ser executado uma vez.

Uma outra maneira de pensar a respeito disso é assim: o loop DoWhile mantém o loop desde que a condição seja True (verdadeira). O loop DoUntil prossegue desde que a condição seja False (falsa).

Fazendo Loop através de uma Collection

O VBA suporta ainda outro tipo de loop – fazer loop em cada objeto em uma coleção de objetos. Lembre-se que uma coleção consiste de uma série de objetos do mesmo tipo. Por exemplo, o Excel tem uma coleção de todas as pastas de trabalho abertas (a coleção Workbooks), e cada pasta de trabalho tem uma coleção de planilhas (a coleção Worksheets).

Quando você precisa fazer loop através de cada objeto em uma coleção, use a estrutura For Each-Next. A sintaxe é

For Each element In *collection*

[*statements*]

[Exit For]

[*statements*]

Next [*element*]

ExcelVBABokks.indb 158

15/02/2013 13:59:46

Capítulo 10: Controlando o Fluxo de Programa e Tomando Decisões 159

O seguinte exemplo faz loop através de cada planilha na pasta de trabalho ativa, e exclui a primeira linha de cada planilha:

```
Sub DeleteRow1()
```

```
Dim WkSht As Worksheet
```

```
For Each WkSht In activeWorkbook.Worksheets
```

```
WkSht.Rows(1).Delete
```

```
Next WkSht
```

```
End Sub
```

Neste exemplo, a variável WkSht (planilha) é um objeto variável que representa cada planilha na pasta de trabalho. Não há nada especial com relação ao nome WkSht — você pode usar qualquer nome de variável que quiser.

O exemplo que segue faz loops através das células em uma faixa e verifica cada célula. O código troca o sinal dos valores (valores negativos tornam-se positivos; valores positivos tornam-se negativos).

Isso é feito multiplicando cada valor por -1. Observe que eu usei uma constru-

ção If-Then, junto com a função IsNumeric VBA, para garantir que a célula contenha um valor numérico:

```
Sub ChangeSign()
```

```
Dim Cell As Range
```

```
For Each Cell In Range("A1:E50")
```

```
    If IsNumeric(Cell.Value) Then
```

```
        Cell.Value = Cell.Value * -1
```

```
    End If
```

```
Next Cell
```

```
End Sub
```

O exemplo de código acima tem um problema: ele altera quaisquer fórmulas na faixa pela qual faz loops através de valores, apagando as suas fórmulas. Provavelmente, não é isso que você quer. Eis uma outra versão da Sub que pula células de fórmula. Ela verifica se a célula tem uma fórmula, acessando a propriedade HasFormula: Sub ChangeSign2()

```
Dim Cell As Range
```

```
For Each Cell In Range("A1:E50")
```

```
    If Not Cell.HasFormula Then
```

```
        If IsNumeric(Cell.Value) Then
```

```
            Cell.Value = Cell.Value * -1
```

```
        End If
```

End If

Next Cell

End Sub

E a seguir está mais um exemplo de For Each-Next. O procedimento faz loops através de cada gráfico em Sheet1 (isto é, cada membro da coleção ChartObjects e altera cada gráfico para uma linha de gráfico.

No exemplo, Cht é uma variável que representa cada ChartObject. Se Sheet1 não tiver ChartObjects, nada acontece.

ExcelVBABokks.indb 159

15/02/2013 13:59:46

160 Parte III: Conceitos de Programação Sub ChangeCharts()

Dim Cht As ChartObject

For Each Cht In Sheets("Plan1").ChartObjects

Cht.Chart.Charttype = xlLine

Next Cht

End Sub

Para escrever um procedimento como ChargeCharts você precisa saber alguma coisa sobre o modelo objeto para gráficos. Essas informações podem ser obtidas, gravando uma macro para descobrir quais objetos estão envolvidos e depois, verificando o sistema de Ajuda para detalhes.

Usuários de Excel 2007 estão com pouca sorte aqui: o gravador de macro em Excel 2007 não grava todas as alterações que você faz no gráfico.

ExcelVBABokks.indb 160

15/02/2013 13:59:46

Capítulo 11

Procedimentos e Eventos

Automáticos



Neste Capítulo

- ☐ Conhecendo os tipos de eventos que podem desencadear uma execução
- ☐ Como descobrir onde colocar o seu código VBA que lida com eventos
- ☐ Executando uma macro quando uma pasta de trabalho está aberta ou fechada
- ☐ Executando uma macro quando uma pasta de trabalho ou planilha está ativada



Você tem uma série de formas de executar um procedimento VBA Sub. Uma delas é organizar para que Sub seja automaticamente

executado. Neste capítulo, abordo os prós e os contras desse recurso potencialmente útil, explicando como acertar as coisas para que uma macro seja executada automaticamente quando ocorrer um evento em especial. Não, este capítulo não é sobre pena de morte.

Preparação para o Grande Evento

Sobre quais tipos de eventos estou falando aqui? Boa pergunta. Basicamente, um *evento* é alguma coisa que acontece no Excel. A seguir, estão alguns exemplos dos tipos de eventos com os quais o Excel pode lidar:

- 5 Uma pasta de trabalho é aberta ou fechada.

- 5 Uma janela é ativada.

- 5 Uma planilha é ativada ou desativada.

5 Dados são fornecidos em uma célula ou a célula é editada.

5 Uma pasta de trabalho é salva.

5 Uma planilha é calculada.

5 Um objeto, tal como um botão, é clicado.

5 Uma tecla em especial, ou uma combinação de teclas é pressionada.

5 Uma hora específica do dia ocorrer.

5 Ocorrência de um erro.

ExcelVBABokks.indb 161

15/02/2013 13:59:46

162 Parte III: Conceitos de Programação A maioria dos programadores de Excel nunca precisa se preocupar com a maioria dos eventos desta lista. No entanto, você deveria pelo menos saber que estes eventos existem, pois algum dia eles podem se tornar úteis. Neste capítulo, eu discuto a maioria dos eventos mais comuns. Para simplificar as coisas, falo sobre dois tipos de eventos: pasta de trabalho e planilha.

A Tabela 11-1 relaciona a maior parte dos eventos relacionados à pasta de trabalho. Se, por algum motivo, você precisar ver a lista completa de eventos relacionados à pasta de trabalho, poderá encontrá-la na Ajuda do sistema.

Tabela 11-1

Eventos de Pasta de Trabalho

Evento

Quando Ele é Acionado

Activate

A pasta de trabalho é ativada.

AddinInstall

Um add-in é instalado (importante apenas para add-ins).

AddinUninstall

O add-in é desinstalado (importante apenas para add-ins).

BeforeClose

A pasta de trabalho é fechada.

BeforePrint

A pasta de trabalho é impressa.

BeforeSave

A pasta de trabalho é salva.

Deactivate

A pasta de trabalho é desativada.

NewSheet

Uma nova planilha é adicionada à pasta de trabalho.

Open

A pasta de trabalho é aberta.

SheetActivate

Uma planilha da pasta de trabalho é ativada.

SheetBeforeDoubleClick Uma célula na pasta de trabalho é clicada duas vezes.

SheetBeforeRightClick

Uma célula na pasta de trabalho é clicada com o botão direito.

SheetCalculate

Uma planilha na pasta de trabalho é recalculada.

SheetChange

Uma alteração é feita em uma célula na pasta de trabalho.

SheetDeactivate

Uma planilha na pasta de trabalho é desativada.

SheetFollowHyperlink

Um hyperlink em uma pasta de trabalho é clicado.

SheetSelectionChange A seleção é alterada.

WindowActivate

A janela da pasta de trabalho é ativada.

WindowDeactivate

A janela da pasta de trabalho é desativada.

WindowResize

A janela da pasta de trabalho é redimensionada.

A Tabela 11-2 relaciona a maioria dos eventos relacionados à planilha.

ExcelVBABokks.indb 162

15/02/2013 13:59:46

Capítulo 11: Procedimentos e Eventos Automáticos 163

Tabela 11-2

Eventos de Planilha

Evento

Quando Ele é Acionado

Activate

A planilha é ativada.

BeforeDoubleClick

Uma célula na planilha é clicada duas vezes.

BeforeRichtClick

Uma célula na planilha é clicada com o botão direito.

Calculate

A planilha é recalculada.

Change

Uma alteração é feita em uma célula na planilha.

Deactivate

A planilha é desativada.

FollowHyperlink

Um hyperlink é ativado.

SelectionChange

A seleção é alterada.

Os eventos são úteis?

A essa altura, você pode estar se perguntando como esses eventos podem ser úteis. Eis um rápido exemplo.

Imagine que você tem uma pasta de trabalho onde você entra com dados na coluna A. O seu chefe diz que precisa saber exatamente quando cada ponto dos dados foi inserido. Entrar com dados é um evento — um evento `WorksheetChange`. Você pode escrever uma macro que responda a esse evento. Aquela macro detecta sempre que

a planilha é alterada. Se a alteração foi feita na coluna A, a macro coloca a data e a hora na coluna B, próximo ao ponto dos dados que você forneceu.

No caso de você ser curioso, eis como tal macro se pareceria (ela deveria estar no módulo Código da planilha). Provavelmente, muito mais simples do que você imaginou que seria, não é?

```
Private Sub Worksheet_Change(ByVal Target As Range)
```

```
If Target.Column = 1 Then
```

```
Target.Offset(0, 1) = Now
```

```
End If
```

```
End Sub
```

Só porque a sua pasta de trabalho contém procedimentos que respondem a eventos não garante que aqueles procedimentos, de fato, roda-CUIDADO!

rão. Como você sabe, é possível abrir uma pasta de trabalho com as macros desativadas. Em tal situação, todas as macros (até procedimentos que respondem a eventos) são desativadas. Tenha isso em mente quando criar pastas de trabalho que se baseiam em procedimentos que lidam com eventos.

ExcelVBABokks.indb 163

15/02/2013 13:59:46

164 Parte III: Conceitos de Programação *Programando procedimentos que lidam com eventos*

Um procedimento VBA executado em resposta a um evento é chamado de um *procedimento que lida com evento*. Esses são sempre procedimentos Sub (em oposto aos procedimentos Function). Escrever esses manejadores de evento é relativamente direto depois que você entende como funciona o processo. Tudo se resume em algumas etapas, as quais explico a seguir: **1. Identifique o evento que deseja para acionar o procedimento.**

2. Pressione Alt + F11 para ativar o Visual Basic Editor.

3. Na Janela de Projeto VBE, clique duas vezes o objeto adequado listado sob Microsoft Excel Objetos.

Para eventos relacionados à pasta de trabalho, o objeto é EstaPasta_de_Trabalho. Para um evento relacionado à planilha, o objeto é Worksheet (tal como Plan1).

4. Na janela Código do objeto, escreva o procedimento que lida com evento que é executado quando o evento ocorre.

Este procedimento terá um nome especial que o identifica como um procedimento que lida com evento.

Estas etapas tornam-se mais claras na medida em que você se adianta no capítulo. Confie em mim.

Aonde Vai o Código VBA?

É muito importante entender aonde vão os seus procedimentos que lidam com evento. Eles precisam ficar na janela Código de um módulo Objeto. Eles não ficam em um módulo VBA padrão. Se você puser o seu procedimento que lida com evento no lugar errado, ele simplesmente não vai funcionar. E você não vê quaisquer mensagens de erro.

A Figura 11-1 mostra a janela VBE com um projeto exibido na janela Projeto (para saber mais sobre o VBE, veja o Capítulo 3). Observe que o projeto VBA para Pasta1 está totalmente expandido e consiste de vários objetos:

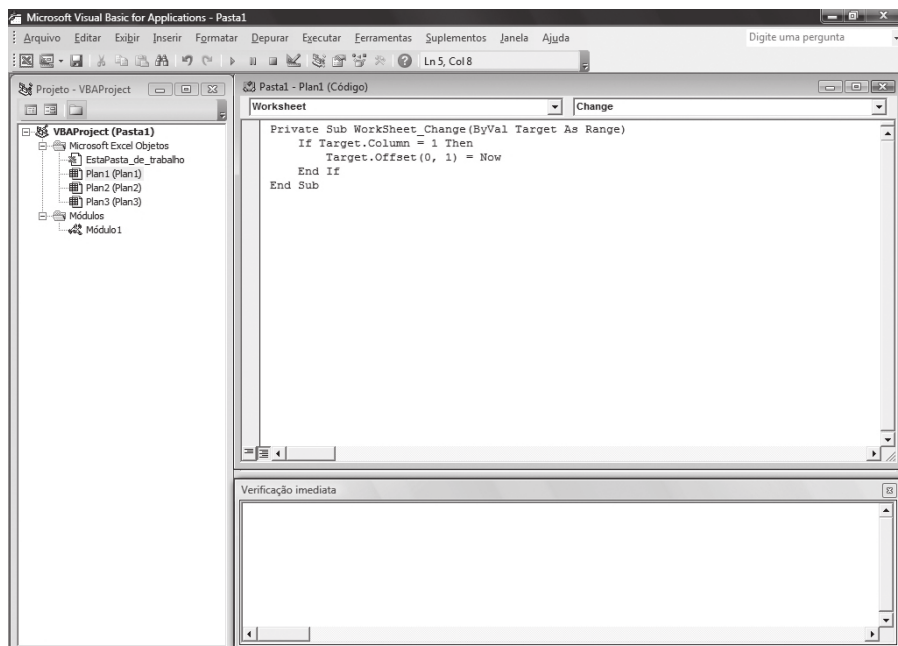
5 Um objeto para cada planilha na pasta de trabalho (neste caso, três objetos Plan1)

5 Um objeto nomeado Estapasta_de_trabalho

5 Um módulo VBA que eu inseri manualmente, usando o comando
Inserir⇒Módulo.

ExcelVBABokks.indb 164

15/02/2013 13:59:46



Capítulo 11: Procedimentos e Eventos Automáticos 165

Figura 11-1:

A janela
VBE exibe
itens para
um único
projeto.

Clicar duas vezes em qualquer um desses objetos exibe o código associado ao item, se houver.

Os procedimentos que lidam com evento que você escreve vão na janela Código para o item EstaPasta_de_Trabalho (para eventos relacionados à pasta de trabalho) ou um dos objetos Plan (para eventos relacionados à planilha).

Na Figura 11-1, a janela Código para o objeto Plan1 é exibida, e por acaso, tem um único procedimento que lida com eventos definido. Viu as duas caixas de listagem drop-down no alto do módulo Código?

Continue lendo para descobrir porque elas são úteis.

Escrevendo um Procedimento

Que Lida com Evento

O VBE o ajuda quando você está pronto para escrever um procedimento que lida com evento; ele exibe uma lista de todos os eventos para o objeto selecionado.

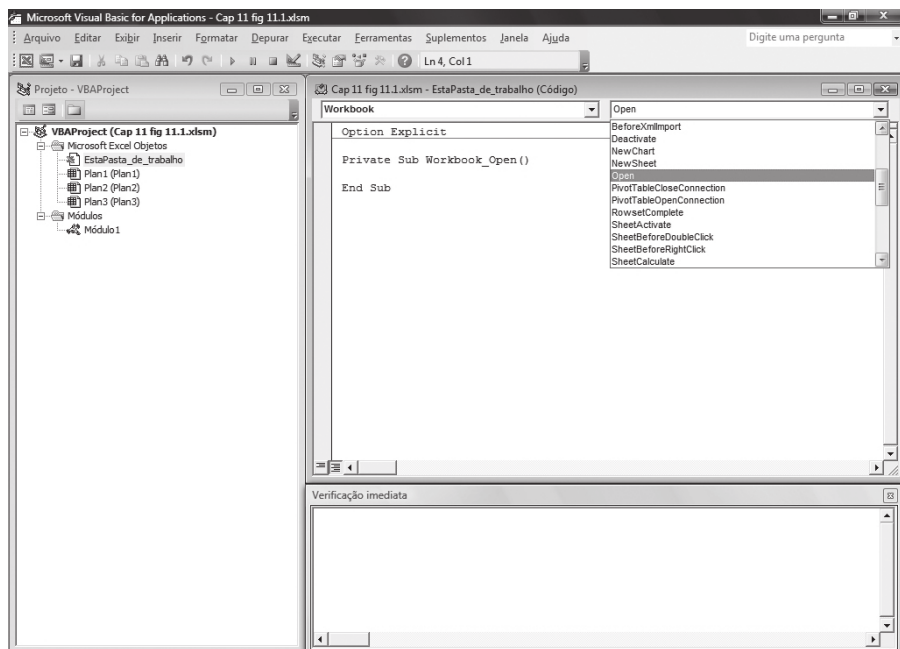
No alto de cada janela Código, você encontra duas listas drop-down: 5 A lista de Objeto (à esquerda)

5 A lista de Procedimento (à direita)

Por padrão, a lista de Objeto na janela Código exibe o termo Geral. Se você estiver escrevendo um procedimento que lida com evento para o objeto EstaPasta_de_Trabalho, precisa escolher Workbook na caixa de listagem Objeto (esta é a única escolha).

ExcelVBABokks.indb 165

15/02/2013 13:59:46



166 Parte III: Conceitos de Programação

Se estiver escrevendo para manipular o evento de um objeto Sheet, você precisa escolher Worksheet (novamente, a única escolha).

Depois de ter feito a sua escolha a partir da lista de Objeto, você pode escolher o evento da lista drop-down Procedimento. A Figura 11-2 mostra as escolhas para um evento relacionado à pasta de trabalho.

Figura 11-2:

Escolhendo
um evento na
janela Código
para o objeto
EstaPasta_
de_trabalho.

Quando você seleciona um evento da lista, automaticamente o VBE começa a criar para você um procedimento que lida com o evento. Esse é um recurso muito útil, pois você pode verificar se os

argumentos adequados são usados.

Eis um pequeno capricho. Quando você selecionou Workbook pela primeira vez na lista Objeto, o VBE supôs que você queria criar um procedimento para o evento Open e o criou para você. Se você estiver, de fato, criando um procedimento Workbook_Open, está ótimo. Mas se estiver criando um procedimento de evento diferente, precisa apagar a Sub Workbook_Open vazia que o Excel criou.

No entanto, a ajuda do VBE só vai até aqui. Ele escreve a declaração Sub e a declaração End Sub. É seu trabalho escrever o código VBA que fica entre essas duas declarações.

Na verdade, você não precisa usar aquelas duas listas drop-down, mas **BRE-SE**

facilita o seu trabalho, pois o nome do procedimento que lida com **LEM**

evento é muito importante. Se você não tiver o nome exato, ele não funciona. Além disso, alguns procedimentos que lidam com evento usam um ou mais argumentos na declaração Sub. Não há como você lembrar quais são aqueles argumentos. Por exemplo, se você selecionar SheetActivate da lista de eventos para um objeto Workbook, o VBE

escreve a seguinte declaração Sub:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
```

ExcelVBABokks.indb 166

15/02/2013 13:59:46

Capítulo 11: Procedimentos e Eventos Automáticos 167

Neste caso, Sh é o argumento passado ao procedimento e é uma variável que representa a planilha na pasta de trabalho ativada. Exemplos neste capítulo esclarecem esta questão.

Exemplos Introdutórios

Nesta seção, apresento alguns exemplos para que você possa ter uma ideia sobre esse negócio de lidar com eventos.

O evento Open para uma pasta de trabalho

Um dos eventos mais usados é o evento Workbook Open. Digamos que você tem uma pasta de trabalho que usa diariamente. O procedimento Workbook_Open neste exemplo é executado cada vez que a pasta de trabalho é aberta. O procedimento verifica o dia da semana; se for Sexta-Feira, o código exibe uma mensagem lembrando-o.

Para criar o procedimento que é executado sempre que ocorrer o evento Workbook Open, siga estes passos:

1. Abra uma pasta de trabalho.

Qualquer pasta de trabalho serve.

2. Pressione Alt + F11 para ativar o VBE.

3. Localize a pasta de trabalho na janela Projeto.

4. Clique duas vezes o nome do projeto para exibir os seus itens, se necessário.

5. Clique duas vezes o item EstaPasta_de_Trabalho.

O VBE exibe uma janela de Código vazia para o objeto

EstaPasta_de_Trabalho.

6. Na janela Código, selecione Workbook a partir da lista drop-down de Objeto (à esquerda).

O VBE insere declarações no início e no final para um procedimento Workbook_Open.

7. Entre com as seguintes declarações, para que o procedimento de evento completo se pareça com isto:

```
Private Sub Workbook_Open()
```

```
Dim Msg As String
```

```
If WeekDay(Now) = 6 Then
```

```
Msg = "Today is Friday. Make sure that you "
```


Msg = Msg & “do your weekly backup!”

MsgBox Msg

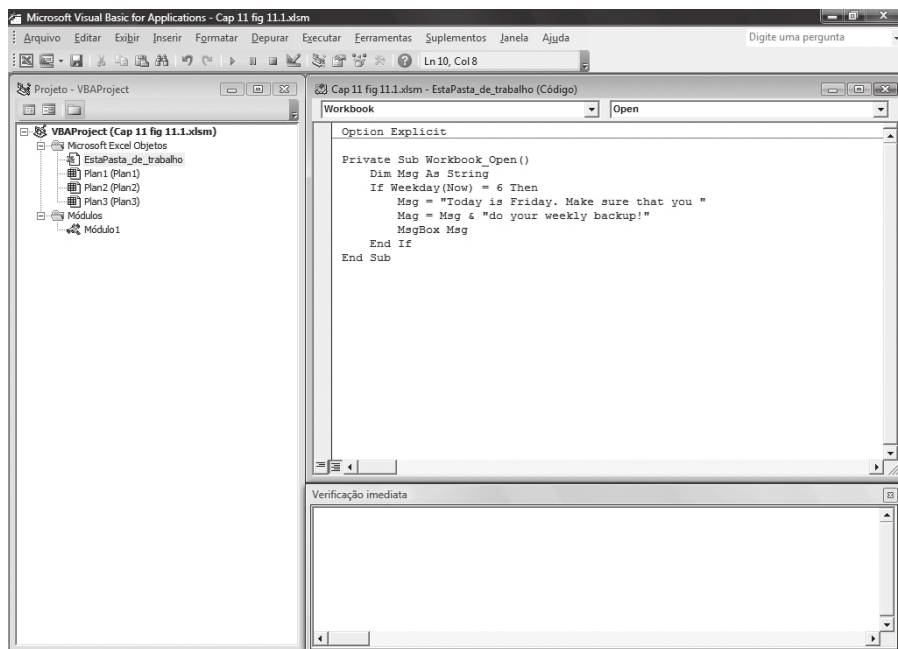
End

If

End Sub

ExcelVBABokks.indb 167

15/02/2013 13:59:47



168 Parte III: Conceitos de Programação

A janela Código deve parecer com a Figura 11-3.

Figura 11-3:

O procedimento para
manipular o

evento é

executado

quando a

pasta de

trabalho é

aberta.

Workbook_Open é automaticamente executado sempre que a pasta de trabalho for aberta. Ele usa a função WeekDay do VBA para determinar o dia da semana. Se for sexta-feira (dia 6), uma caixa de mensagem lembra o usuário para fazer o backup semanal. Se não for sexta-feira, nada acontece.

Se hoje não for sexta-feira, você pode ter dificuldades para testar esse procedimento. Aqui está uma oportunidade de testar a sua habilidade em VBA. Você pode modificar esse procedimento como quiser. Por exemplo, a seguinte versão exibe uma mensagem sempre que a pasta de trabalho é aberta. Isso é irritante depois de algum tempo, acredite em mim.

```
Private Sub workbook_Open()
```

```
Msg = “Esta é a pasta de trabalho legal de Frank!”
```

```
MsgBox Msg
```

```
End Sub
```

Um procedimento Workbook_Open pode fazer quase qualquer coisa.

Geralmente, esses acionadores de evento são usados para o seguinte: 5 Exibir mensagens de boas vindas (tal como na bacana pasta de trabalho do Frank).

5 Abrir outras pastas de trabalho.

5 Ativar uma planilha em especial na pasta de trabalho.

5 Configurar menus de atalho personalizados.

ExcelVBABokks.indb 168

15/02/2013 13:59:47

Capítulo 11: Procedimentos e Eventos Automáticos 169

O evento BeforeClose para

uma pasta de trabalho

Eis um exemplo do procedimento que lida com eventos Workbook_BeforeClose, o qual é executado imediatamente antes da pasta de trabalho ser fechada. Esse procedimento está localizado na janela Código para um objeto This Workbook (Esta Pasta_de_Trabalho): Private Sub Workbook_BeforeClose(Cancel As Boolean)

Dim Msg As String

Dim Ans As Integer

Dim FName As String

Msg = “Gostaria de fazer um backup deste arquivo?”

Ans = MsgBox(Msg, vbYesNo)

If Ans = vbYes Then

FName = “F:\BACKUP\” & ThisWorkbook.Name

ThisWorkbook.SaveCopyAs FName

End If

End Sub

Esta rotina usa uma caixa de mensagem para perguntar ao usuário se

ele gostaria de fazer uma cópia da pasta de trabalho. Se a resposta for sim, o código usa o método SaveCopyAs, para salvar uma cópia do arquivo no drive F. Se você adaptar esse procedimento para o seu próprio uso, provavelmente precisará mudar o drive e o caminho.

Frequentemente, programadores de Excel usam um procedimento

Workbook_BeforeClose para fazer uma limpeza. Por exemplo, se você usa um procedimento Workbook_Open para mudar algumas configurações ao abrir uma pasta de trabalho (ocultando a barra de status, por exemplo), é simplesmente adequado que você retorne as configurações à posição original quando fechar a pasta de trabalho. Você pode realizar essa limpeza eletrônica com um procedimento Workbook_BeforeClose.

Existe uma advertência com o evento Workbook_BeforeClose. Se você **CUIDADO!**

fechar o Excel e qualquer arquivo aberto tiver sido modificado desde a última vez em que foi salvo, o Excel mostrará a sua habitual caixa de mensagem “Deseja salvar as alterações”. Clicar o botão Cancelar cancela todo o processo de encerramento. Mas, o evento Workbook_BeforeClose terá sido executado de qualquer forma.

O evento BeforeSave para

uma pasta de trabalho

O evento BeforeSave, como o nome sugere, é disparado antes que uma pasta de trabalho seja salva. Esse evento acontece quando você usa o comando Arquivo⇨Salvar ou o comando Arquivo⇨Salvar Como.

O seguinte procedimento a seguir que é colocado na janela Código de um objeto This Workbook, demonstra o evento BeforeSave. A rotina atualiza o valor em uma célula (célula A1 em Sheet1) sempre que a pasta de trabalho é salva. Em outras palavras, a célula A1 serve como um contador, para controlar o número de vezes que o arquivo foi salvo.

ExcelVBABokks.indb 169

15/02/2013 13:59:47

170 Parte III: Conceitos de Programação Private Sub
Workbook_BeforeSave(ByVal SaveAsUI _

As Boolean, Cancel as Boolean)

```
Sheets("Sheet1").Range("A1").Value = _
```

```
Sheets("Sheet1").Range("A1").Value
```

```
+ 1
```

```
End Sub
```

Observe que o procedimento `Workbook_BeforeSave` tem dois argumentos, `SaveAsUI` (Salvar Como Interface de Usuário) e `Cancel` (Cancelar).

Para demonstrar como estes argumentos funcionam, examine a seguinte macro, que é executada antes que a pasta de trabalho seja salva. Este procedimento evita que o usuário salve a pasta de trabalho com um nome diferente. Se o usuário escolher o comando Arquivo>Salvar Como, então o argumento `SaveAsUI` é Verdadeiro.

Quando o código executar, ele verifica o valor `SaveAsUI`. Se essa variável for `True` (verdadeira), o procedimento exibe uma mensagem e configura `Cancel` para `True`, o que cancela a operação `Save`.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI _
```

```
As Boolean, Cancel As Boolean)
```

```
If SaveAsUI Then
```

```
MsgBox "Você não pode salvar uma cópia desta  
pasta!"
```

```
Cancel = True
```

```
End If
```

```
End Sub
```

Tenha em mente que este procedimento, na verdade, não evita que alguém salve uma cópia com um nome diferente. Se alguém quiser, de fato, fazê-lo, basta abrir a pasta de trabalho com as macros desativadas. Quando as macros estão desativadas, os procedimentos acionadores de eventos também estão — o que faz sentido, pois afinal, todos eles são macros.

Exemplos de Ativação de Eventos

Uma outra categoria de eventos consiste em ativar e desativar objetos — especificamente planilhas e pastas de trabalho.

Ativar e desativar eventos em uma planilha

O Excel pode detectar quando uma planilha em especial é ativada ou desativada e executar uma macro quando ocorrer qualquer um desses eventos. Esses procedimentos que lidam com eventos ficam na janela Código do objeto Sheet.

ExcelVBABokks.indb 170

15/02/2013 13:59:47

Capítulo 11: Procedimentos e Eventos Automáticos 171

Você pode acessar rapidamente a janela de código de uma planilha, clicando com o botão direito na guia da planilha e selecionando Ver Código.

O exemplo a seguir mostra um simples procedimento que é executado sempre que uma planilha em especial é ativada. Esse código apenas faz surgir uma caixa de mensagem que exibe o nome da planilha ativa: `Private Sub Worksheet_Activate() MsgBox "You just activated " & ActiveSheet.Name`

`End Sub`

Eis um outro exemplo que ativa a célula A1 sempre que a planilha for ativada:

```
Private Sub Worksheet_Activate()
```

```
Range("A1").Activate
```

End Sub

Ainda que o código nestes dois procedimentos seja quase tão simples quanto possível, os procedimentos que acionam eventos podem ser tão complexos quanto você quiser.

O seguinte procedimento (o qual está armazenado na janela Código do objeto Sheet1) usa o evento Deactivate (Desativar) para evitar que um usuário ative qualquer outra planilha na pasta de trabalho. Se Sheet1

estiver desativada (isto é, outra planilha estiver ativa), o usuário recebe uma mensagem e Sheet1 é ativada.

```
Private Sub Worksheet_Deactivate()
```

```
MsgBox "You must stay on Sheet1"
```

```
Sheets("Sheet1").Activate
```

```
End Sub
```

A propósito, eu não recomendo usar procedimentos como este, que tenta assumir o Excel. Esses aplicativos chamados de “ditadores”

podem ser bem frustrantes e confusos ao usuário. Ao invés disso, recomendo treinar o usuário a usar corretamente o seu aplicativo.

Ativar e desativar eventos

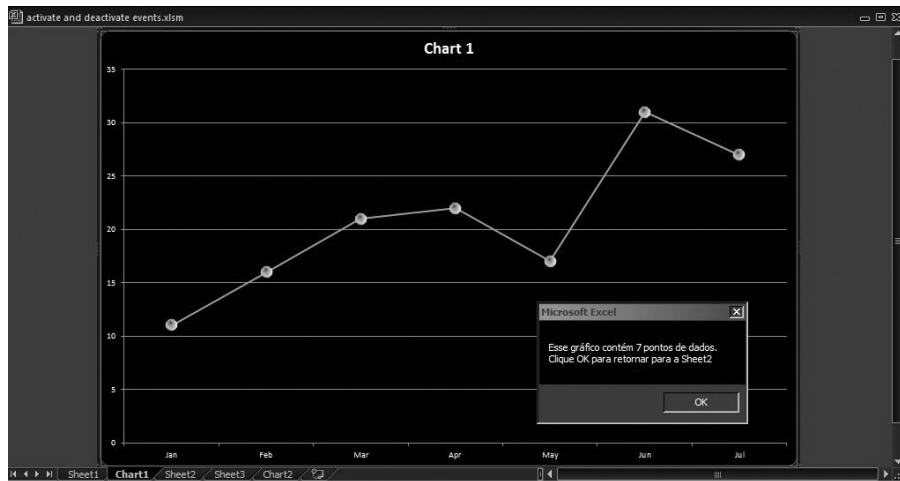
em uma pasta de trabalho

Os exemplos anteriores usam eventos associados a uma planilha específica. O objeto EstaPasta_de_Trabalho também aciona eventos que lidam com a ativação e desativação de planilha. O seguinte procedimento, o qual é armazenado na janela Código do objeto ThisWorkbook, é executado quando *qualquer* planilha na pasta de trabalho é ativada. O

código exibe uma mensagem com o nome da planilha ativada.

ExcelVBABokks.indb 171

15/02/2013 13:59:47



172 Parte III: Conceitos de Programação

Private Sub Workbook_SheetActivate(ByVal Sh As Object)

MsgBox Sh.Name

End Sub

O procedimento `Workbook_SheetActivate` usa o argumento `Sh`. `Sh` é uma variável que representa o objeto ativo `Sheet`. A caixa de mensagem exibe a propriedade `Name` do objeto `Sheet`. O próximo exemplo está contido em uma janela do código de `EstaPasta_de_Trabalho`. Ele consiste de dois procedimentos que lidam com eventos:

5 Workbook_SheetDeactivate: Executado quando qualquer

planilha na pasta de trabalho for desativada. Ele armazena a planilha que é desativada em um objeto variável (a palavra chave Set cria um objeto variável).

5 Workbook_SheetActivate: Executado quando qualquer planilha

na pasta de trabalho for ativada. Ele verifica o tipo de planilha que está ativa (usando a função TypeName). Se a planilha for

uma planilha de gráfico, o usuário recebe uma mensagem (ver

Figura 11-4). Quando o botão OK na caixa de mensagem for

clicado, a planilha *anterior* (que está armazenada na variável OldSheet) é reativada.

Figura

11-4:

Quando

uma

planilha de

gráfico é

ativada, o

usuário vê

uma

mensagem

como esta.

ExcelVBABokks.indb 172

15/02/2013 13:59:47

Capítulo 11: Procedimentos e Eventos Automáticos 173

Uma pasta de trabalho que contém este código está disponível no site deste livro.

```
Dim OldSheet As Object
```

```
Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
```

```
Set OldSheet = Sh
```

```
End Sub
```

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
```

```
Dim Msg As String
```

```
If TypeName(Sh) + "Chart" Then
```

```
Msg = "Esse gráfico contém "
```

```
Msg = Msg & ActiveChart.SeriesCollection(1).
```

```
Points.Count
```

```
Msg = Msg & " pontas de dados." & vbNewLine
```

```
Msg = Msg & "Clique OK para retornar a " & OldSheet.
```

```
Name
```

```
MsgBox Msg
```

```
OldSheet.Activate
```

```
End If
```

```
End Sub
```

Eventos de ativação de pasta de trabalho

O Excel também reconhece o evento que ocorre quando você ativa ou desativa uma pasta de trabalho em especial. O seguinte código, que

está contido na janela Código do objeto EstaPasta_de_Trabalho, é executado sempre que a pasta de trabalho é ativada. O procedimento apenas maximiza a janela da pasta de trabalho.

```
Private Sub Workbook_Activate()
```

```
ActivateWindow.WindowState = xlMaximized
```

```
End Sub
```

O código Workbook_Deactivate, mostrado a seguir, é executado quando uma pasta de trabalho é desativada. Eis um exemplo de procedimento que copia a faixa selecionada. Ele pode ser útil se você estiver copiando dados de muitas áreas diferentes e colando-os em uma pasta de trabalho diferente. Selecione a faixa, ative a outra pasta de trabalho, selecione o destino e pressione Ctrl + V para colar os dados copiados.

```
Private Sub Workbook_Deactivate()
```

```
ThisWorkbook.Windows(1).RangeSelection.Copy
```

```
End Sub
```

Simples assim, este procedimento exigiu algumas tentativas antes que eu conseguisse fazê-lo funcionar corretamente. Primeiro, eu tentei isto: Selection.copy

ExcelVBABokks.indb 173

15/02/2013 13:59:47

174 Parte III: Conceitos de Programação Esta declaração não funcionou conforme o pretendido. Ela copiou a faixa da segunda pasta de trabalho (a que eu ativei depois de desativar a primeira pasta de trabalho). Isso, porque a segunda pasta de trabalho tornou-se a primeira pasta de trabalho depois da ocorrência do evento de desativação.

Esta declaração também não funcionou. Na verdade, ela me deu um erro em tempo de execução:

```
ThisWorkbook.ActiveSheet.Selection.Copy
```

Eventualmente, eu me lembrei da propriedade RangeSelection de um objeto Window. E essa fez o truque.

Worksheet (Planilha)

Na seção anterior, apresentei exemplos de eventos de ativação e desativação de planilha. Nesta seção, eu discuto três eventos adicionais que ocorrem em planilhas: clicar duas vezes em uma célula, clicar com o botão direito em uma célula e alterar uma célula.

O evento BeforeDoubleClick

Você pode configurar um procedimento VBA para ser executado

quando o usuário clicar duas vezes em uma célula. No exemplo a seguir (o qual é armazenado na janela Código de um objeto Sheet), clicar duas vezes em uma célula da planilha deixa a célula em negrito (se ela não estiver em negrito) ou sem negrito (se ela estiver em negrito).

```
Private Sub Worksheet_BeforeDoubleClick _  
(ByVal Target As Excel.Range, Cancel As Boolean)
```

```
Target.Font.Bold = Not Target.Font.Bold
```

```
Cancel = True
```

```
End Sub
```

O procedimento `Worksheet_BeforeDoubleClick` tem dois argumentos: `Target` e `Cancel`. `Target` representa a célula (um objeto `Range`) que foi clicado duas vezes. Se `Cancel` estiver configurado para `Verdadeiro`, a ação padrão de clicar duas vezes não acontece.

A ação padrão de clicar duas vezes em uma célula é colocar o Excel no modo de edição da célula. Eu não quero que isso aconteça, portanto, configuro `Cancel` para `Verdadeiro`.

O evento BeforeRightClick

O evento `BeforeRightClick` é semelhante ao evento `BeforeDoubleClick`, exceto que ele consiste em clicar com o botão direito em uma célula. O

Capítulo 11: Procedimentos e Eventos Automáticos 175

seguinte procedimento verifica que a célula que foi clicada com o botão direito contém um valor numérico. Se for o caso, o código exibe a caixa de diálogo Format Number e configura o argumento Cancel para Verdadeiro (evitando a exibição normal do menu de atalho). Se a célula não tiver um valor numérico, nada de especial acontece – o menu de atalho é exibido, como de costume.

```
Private Sub Worksheet_BeforeRightClick _
```

```
(ByVal Target As Excel.Range, Cancel As Boolean)
```

```
If IsNumeric(Target) And Not IsEmpty(Target) Then
```

```
Application.CommandBars.ExecuteMso
```

```
-
```

```
("NumberFormatsDialog")
```

```
Cancel = True
```

```
End If
```

```
End Sub
```

Observe que o código, que está disponível no site deste livro, faz uma verificação adicional para ver se a célula não está vazia. Isso, porque o VBA considera células vazias como sendo numéricas. Não me pergunte o motivo; ele simplesmente considera.

O evento Change

O evento Change acontece sempre que qualquer célula na planilha é alterada. No seguinte exemplo, o procedimento Worksheet_Change evita, efetivamente, que um usuário entre com um valor não numérico na célula A1. Este código é armazenado na janela Código de um objeto Sheet.

```
Private Sub Worksheet_Change(ByVal Target As Range)
```

```
If Target.Address = "$A$1" Then
```

```
If Not IsNumeric(Target) Then
```

MsgBox = “Insira um número na célula A1.”

Range(“A1”).ClearContents

Range(“A1”).Activate

End

If

End If

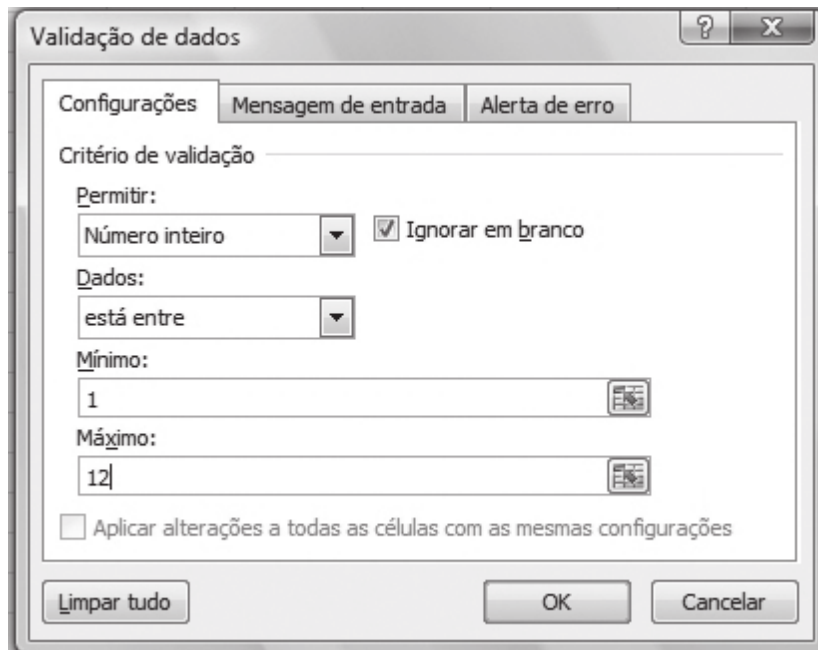
End Sub

O único argumento para o procedimento Worksheet_Change representa a faixa que foi alterada. A primeira declaração verifica se o endereço da célula é \$A\$1. Se for, o código usa a função IsNumeric para determinar se a célula contém um valor numérico. Se não, aparece uma mensagem e o valor da célula é apagado. Então, a célula A1 é ativada – útil, se o indicador da célula mudou para uma célula diferente depois da entrada ser feita. Se houver a alteração em qualquer célula, exceto A1, nada acontece.

Por que não usar validação de dados?

Você pode estar familiarizado com o comando Dados\Ferramentas de Dados\Validação de Dados. Esse é um recurso conveniente que visa garantir que apenas dados do tipo adequado sejam inseridos em uma ExcelVBABook.indb 175

15/02/2013 13:59:47



176 Parte III: Conceitos de Programação

célula ou faixa em especial. Ainda que o comando Dados⇨Ferramentas de Dados⇨Validação de Dados seja útil, definitivamente, ele não é à prova de tolos. Para demonstrar, inicie com uma planilha em branco e execute as seguintes etapas: **1. Selecione a faixa A1:C12.**

2. Escolha Dados⇨Ferramentas de Dados⇨Validação de Dados.

3. Configure os seus critérios de validação para só aceitar números inteiros entre 1 e 12, conforme mostrado na Figura 11-5.

Figura 11-5:

Estas

configura-

ções só

permitem

números

inteiros entre

1 e 12.

Agora, entre com alguns valores na faixa A1:C12. A validação de dados funciona como deveria. Porém, para vê-la se desfazer nas costuras, tente isso:

1. Entre com -1 em qualquer célula fora da faixa de validação (qualquer célula fora de A1:C12).

2. Pressione Ctrl + C para copiar o número negativo para a Clipboard (Área de Transferência).

3. Selecione qualquer célula na faixa de validação.

4. Pressione Ctrl + V para colar o valor copiado.

Você descobre que a operação de colar é permitida. Porém, olhe um pouco mais de perto e você descobre que a célula na qual você colou o valor negativo não tem mais qualquer critério de validação. Colar apaga os critérios de validação de dados! O rigor para tratar essa falha depende do seu aplicativo. Na próxima seção, descrevo como usar o evento Change para fornecer uma validação melhor.

Colar apaga a validação de dados porque o Excel considera a validação um formato de célula. Portanto, ela está na mesma classificação que **DE ESPECIALISTA**

O

tamanho de fonte, cor ou outros atributos semelhantes. Quando você **P A P**

cola uma célula, está substituindo os formatos da célula. Infelizmente, tais formatos também incluem as regras de validação.

ExcelVBABokks.indb 176

15/02/2013 13:59:48

Capítulo 11: Procedimentos e Eventos Automáticos 177

Como evitar que a validação de dados seja eliminada

O procedimento nesta seção demonstra como evitar que os usuários

copiem dados e apaguem regras de validação de dados. Este exemplo supõe que a planilha tenha uma faixa chamada InputRange e que essa área de entrada contenha regras de validação de dados (configuradas, usando o comando Dados⇨Ferramentas de Dados⇨Validação de Dados). A faixa pode ter quaisquer regras de validação que você quiser.

Uma pasta de trabalho contendo este código está disponível no site deste livro:

```
Private Sub Worksheet_Change(ByVal Target As Range)
```

```
Dim VT As Long
```

```
‘Todas as células na faixa de validação
```

```
‘ainda?
```

```
On Error Resume Next
```

```
VT = Range(“InputRange”).Validation.Type
```

```
If Err.Number < > 0 Then
```

```
Application.Undo
```

```
MsgBox “Sua última operação foi cancelada.” & _
```

```
“Teria deletado as regras de validação de dados.”, _
```

```
vbCritical
```

```
End If
```

```
End Sub
```

O procedimento é executado sempre que uma célula é alterada. Ele verifica o tipo de validação da faixa (chamada de InputRange) que *deveria* conter as regras de validação de dados. Se a variável VT

contiver um erro, isso significa que uma ou mais células na InputRange não têm mais validação de dados (provavelmente, o usuário copiou alguns dados sobre ela). Se esse for o caso, o código executa o método Undo (desfazer) do objeto Application e reverte a ação do usuário.

Depois, ele exibe uma caixa de mensagem.

Efeito em cadeia? É impossível apagar as regras de validação copiando dados. Quando o Excel apresentar falhas, use o VBA para repará-lo.

Eventos Não Associados a Objetos

Os eventos que discuti anteriormente neste capítulo são associados ou a um objeto Workbook ou a um objeto Worksheet. Nesta seção, discuto dois tipos de eventos que não estão associados a objetos: horário e toques de tecla.

Como horário e toques de tecla não estão associados a um objeto **BRESE**

específico, tal como uma pasta de trabalho ou uma planilha, você **LEM** programa esses eventos em um módulo normal VBA (diferente de outros eventos discutidos neste capítulo).

ExcelVBABokks.indb 177

15/02/2013 13:59:48

178 Parte III: Conceitos de Programação *O evento OnTime*

O evento OnTime acontece com a ocorrência de uma hora específica do dia. O exemplo a seguir demonstra como fazer o Excel executar um procedimento quando ocorre o evento 15h00. Nesse caso, uma voz robotizada diz para você acordar, acompanhada por uma caixa de diálogo: Sub SetAlarm()

Application.OnTime 0.625, "DisplayAlarm"

End Sub

Sub DisplayAlarm()

Application.Speech.Speak ("Hey, acorde")

MsgBox " É hora para seu descanso vespertino!"

End Sub

Neste exemplo, eu uso o método OnTime do objeto Application. Este método toma dois argumentos: a hora (0.625 ou 15h00) e o nome do procedimento Sub para executar quando ocorrer o evento da hora (DisplayAlarm).

Este procedimento é bem útil se você tem a tendência de ficar tão envolvido em seu trabalho que esquece de reuniões e encontros. Basta ajustar um evento OnTime para se lembrar.

A maioria das pessoas (inclusive este autor) encontra dificuldades em **DICA**

pensar em horário em termos do sistema de numeração do Excel.

Portanto, você pode querer usar a função TimeValue (Valor de Hora) do VBA para representar a hora. TimeValue converte uma string que se parece com uma hora em um valor com o qual o Excel pode lidar. A seguinte declaração mostra uma maneira fácil de programar um evento para as 15 horas: Application.OnTime TimeValue(“3:00:00 pm”), “DisplayAlarm”

Se você quiser programar um evento referente ao horário atual – por exemplo, 20 minutos a contar de agora – pode usar uma declaração como esta: Application.OnTime Now + TimeValue(“00:20:00”), “DisplayAlarm”

Também é possível usar o método OnTime para rodar um procedimento VBA em determinado dia. Você precisa assegurar que o seu computador continue rodando e que a pasta de trabalho com o procedimento seja mantida aberta. A seguinte declaração executa o procedimento DisplayAlarm às 17h00, em 31 de Dezembro de 2010: Application.OnTime DateValue(“12/31/2010 5:00 pm”), _
“DisplayAlarm”

ExcelVBABokks.indb 178

15/02/2013 13:59:48

Capítulo 11: Procedimentos e Eventos Automáticos 179

Esta linha de código em especial pode ser útil para avisá-lo de que precisa ir para casa e se preparar para as festas do Ano Novo.

Eis outro exemplo que usa o evento OnTime. Executar os procedimentos UpdateClock escreve a hora na célula A1 e também programa um outro evento cinco segundos depois. Esse evento roda novamente o procedimento UpdateClock. O efeito em cadeia ocorre porque a célula A1 é atualizada com o horário atual a cada cinco

segundos. Para interromper os eventos, execute o procedimento StopClock (que pode cancelar o evento). Veja que NextTick é uma variável ao nível de módulo que armazena o horário para o próximo evento.

```
Dim NextTick As Date
```

```
Sub UpdateClock()
```

```
‘ Atualiza a célula A1 com a hora atual
```

```
ThisWorkbook.Sheets(1).Range(“A1”) = Time
```

```
‘ Configura o próximo evento para
```

```
NextTick = Now + TimeValue(“00:00:05”)
```

```
Application.OnTime NextTick, “UpdateClock”
```

```
End Sub
```

```
Sub StopClock()
```

```
‘ Cancela o evento atual (para o relógio)
```

```
On Error Resume Next
```

```
Application.OnTime NextTick, “UpdateClock”, , False
```

```
End Sub
```

O evento OnTime prossegue mesmo depois de a pasta de trabalho estar fechada. Em outras palavras, se você fechar a pasta de trabalho sem **CUIDADO!**

rodar o procedimento StopClock, a pasta de trabalho se abrirá por si própria em cinco segundos (supondo que o Excel ainda estava rodando). Para evitar isso, use um procedimento de evento Workbook_BeforeClose, que contém a seguinte declaração: Call StopClock

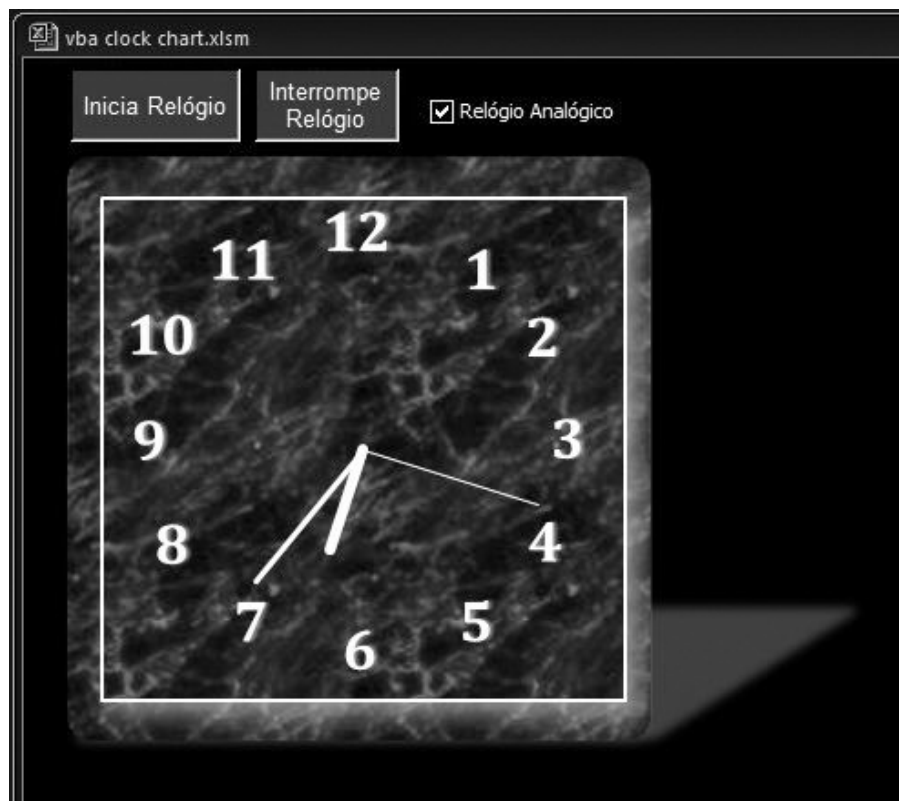
O método OnTime tem dois argumentos adicionais. Se você pretende usar esse método, deve consultar a ajuda online quanto a detalhes completos.

Se você quiser ver um aplicativo bem complicado, faça um download de uma cópia da minha pasta de trabalho de relógio analógico,

mostrado na Figura 11-6. O rosto do relógio, na verdade, é um gráfico atualizado a cada segundo, para exibir a hora do dia. Inútil, mas divertido.

ExcelVBABokks.indb 179

15/02/2013 13:59:48



180 Parte III: Conceitos de Programação

Figura

11-6:

Aplicativo

do meu

relógio

analógico.

Eventos de pressionamento de teclas

Enquanto você trabalha, o Excel monitora constantemente o que você digita. Por causa disso, você pode configurar um pressionamento de tecla ou uma combinação de teclas para executar um procedimento.

Eis um exemplo que reatribui as teclas PgDn (Page Down e PgUp (Page Up =): Sub Setup_OnKey()

```
Application.OnKey "{PgDn}", "PgDn_Sub"
```

```
Application.OnKey "{PgUp}", "PgUp_Sub"
```

```
End Sub
```

```
Sub PgDn_Sub()
```

```
On Error Resume Next
```

```
ActivateCell.Offset(1, 0).Activate
```

```
End Sub
```

```
Sub PgUp_Sub()
```

```
On Error Resume Next
```

```
ActivateCell.Offset(-1, 0).Activate
```

```
End Sub
```

Depois de configurar os eventos OnKey, executando o procedimento Setup_OnKey, pressionar PgDn o move uma linha para baixo. Pressionar PgUp o move uma linha para cima.

Note que os códigos de tecla estão entre chaves, não entre parênteses.

Para uma relação completa de códigos de teclado, consulte o sistema de Ajuda. Procure por *OnKey*.

ExcelVBABokks.indb 180

15/02/2013 13:59:48

Neste exemplo, eu uso On Error Resume Next para ignorar quaisquer erros que sejam gerados. Por exemplo, se a célula ativa estiver na primeira linha, tentar mover uma linha para cima causa um erro que pode ser ignorado com segurança. E se uma planilha de gráfico estiver ativa, não há célula ativa.

Executando a seguinte rotina, você cancela os eventos OnKey:

```
Sub Cancel_OnKey()  
  
Application.OnKey "{PgDn}"  
  
Application.OnKey "{PgUp}"  
  
End Sub
```

Usar uma string vazia como o segundo argumento para o método

OnKey *não* cancela o evento OnKey. Ao contrário, ele leva o Excel a **CUIDADO!**

simplesmente ignorar o toque de teclado. Por exemplo, a seguinte declaração diz ao Excel para ignorar Alt + F4. O sinal de porcentagem representa a tecla Alt:

```
Application.OnKey "%{F4}", " "
```

Embora você possa usar o método OnKey para designar uma tecla de **DICA**

atalho para executar uma macro, você deveria usar a caixa de diálogo Macro Options (Opções de Macro) para essa tarefa. Para mais detalhes, veja o Capítulo 5.

Se você fechar a pasta de trabalho que contém o código e deixar o **BRE-SE**

Excel aberto, o método OnKey não será reiniciado. Como consequência, **LEM**

pressionar a tecla de atalho levará o Excel a abrir, automaticamente, o arquivo com a macro. Para evitar que isso aconteça, você deve complementar o código de evento Workbook_BeforeClose (mostrado anteriormente neste capítulo), para reiniciar o evento OnKey.

Capítulo 12

Técnicas de Tratamento

de Erros



Neste Capítulo

- ☐ Como entender a diferença entre erros de programação e erros de tempo de execução
- ☐ Armadilhas e manipulação de erros de tempo de execução
- ☐ Como usar o VBA nas declarações Error e Resume
- ☐ Descobrindo como você pode usar um erro em seu benefício



Errar é humano. Prever erros é divino. Ao trabalhar com VBA, você deve estar ciente de duas amplas classes de erros: erros de programação e *erros em tempo de execução*. Eu abordo erros de programação, também conhecidos como *bugs*, no Capítulo 13. Um programa bem escrito lida com erros da maneira como Fred Astaire dançava: graciosamente. Felizmente, o VBA inclui diversas ferramentas para ajudá-lo a identificar erros — e depois, a lidar graciosamente com eles.

Tipos de Erros

Se você já experimentou qualquer um dos exemplos neste livro, provavelmente encontrou uma ou mais mensagens de erro. Alguns desses erros resultam de código VBA ruim. Por exemplo, você pode escrever incorretamente uma palavra-chave ou digitar uma declaração com a sintaxe errada. Se cometer tal erro, você não será capaz de executar o procedimento até corrigi-lo.

Este capítulo não trata desses tipos de erros. Ao invés, discuto erros em tempo de execução — os erros que acontecem enquanto o Excel executa o seu código VBA. Mais especificamente, este capítulo cobre os seguintes tópicos fascinantes:

5 Identificação de erros

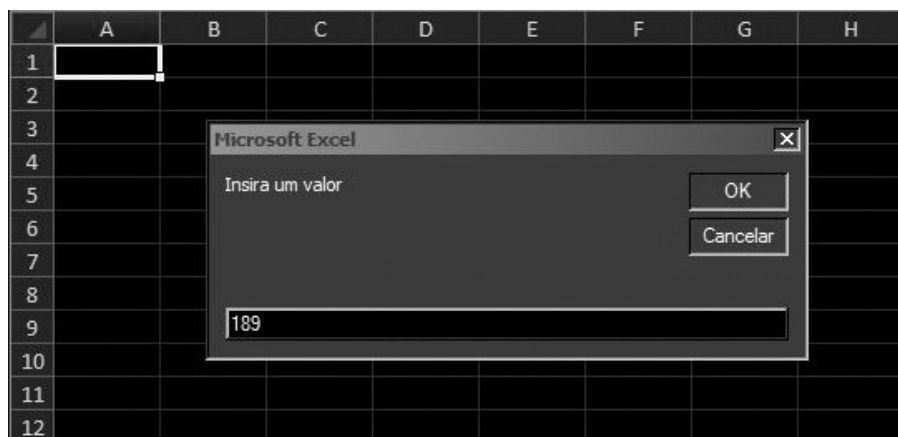
5 Fazer alguma coisa quanto ao erro que acontece

5 Recuperar-se dos erros

5 Criar erros intencionais (sim, às vezes um erro pode ser uma coisa boa).

ExcelVBABokks.indb 183

15/02/2013 13:59:48



184 Parte III: Conceitos de Programação

O objetivo final de lidar com erros é escrever código que evite, tanto quanto possível, exibir mensagens de erro do Excel. Em outras palavras, você quer prever erros em potencial e lidar com eles antes que o Excel tenha uma chance de levantar a sua cara feia com uma mensagem de erro (normalmente) pouco informativa.

Um Exemplo Errôneo

Para começar, eu desenvolvi uma macro VBA curta. Ative o VBE, insira um módulo e entre com o seguinte código:

```
Sub EnterSquareRoot()
```

```
Dim Num As Double
```

```
‘ Tela para valor
```

```
Num = InputBox(“Insira um valor”)
```

```
‘ Insira a raiz quadrada
```

```
ActiveCell.Value = Sqr(Num)
```

```
End Sub
```

Conforme mostrado na Figura 12-1, este procedimento pede um valor ao usuário. Depois, ele executa um cálculo mágico e entra com a raiz quadrada daquele valor na célula ativa.

Figura 12-1:

A função

InputBox

(Caixa de

Entrada)

exibe uma

caixa de

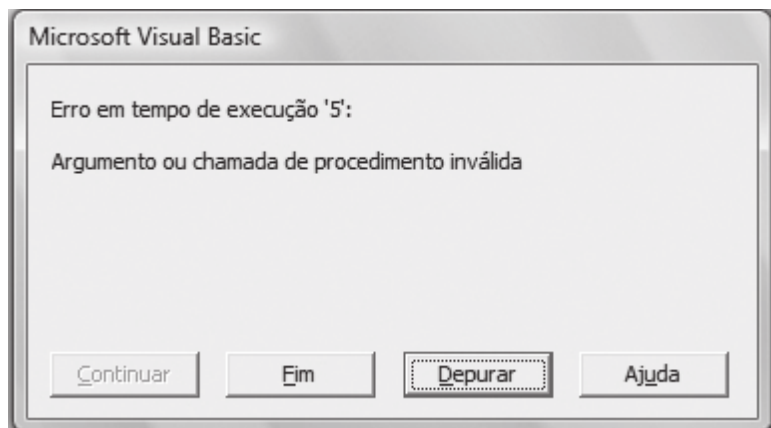
diálogo que

pede um

valor ao

usuário.

Você pode executar este procedimento diretamente do VBE, pressionando F5. Alternativamente, você pode acrescentar um botão a uma planilha (use Desenvolvedor > Controles > Inserir e selecione o botão nos Controles de Formulário para fazer isso) e, depois, atribua a macro ao botão (o Excel pede que você designe a macro). Então, é possível rodar o procedimento simplesmente clicando no botão.



Capítulo 12: Técnicas de Tratamento de Erros 185

A imperfeição da macro

Execute o código umas duas vezes para experimentá-lo. Ele funciona muito bem, não é? Agora, tente entrar com um número negativo quando lhe for pedido um valor. Opa! Neste planeta, tentar calcular a raiz quadrada de um número negativo é ilegal. O Excel responde com a mensagem mostrada na Figura 12-2, indicando que o seu código gerou um erro em tempo de execução. Por ora, apenas clique o botão Fim. Se você clicar o botão Depurar, o Excel suspende a macro, assim você pode usar as ferramentas de depuração que ajudam a rastrear o erro (eu descrevo as ferramentas de depuração no Capítulo 13).

Figura 12-2:

O Excel

exibe esta

mensagem

de erro

quando o

procedimento tenta

calcular

a raiz

quadrada

de um

número

negativo.

A maioria das pessoas não acha as mensagens de erro do Excel muito úteis (por exemplo, procedimento ou argumento inválido). Para aperfei-

çoar o procedimento, você precisa prever o erro e lidar com ele mais graciosamente. Em outras palavras, precisa acrescentar algum código para lidar com erro.

Eis uma versão modificada de EnterSquareRoot:

```
Sub EnterSquareRoot2()
```

```
Dim Num As Double
```

```
‘ Prompt for a value
```

```
Num = InputBox(“Insira um valor”)
```

```
‘ Make sure the number is nonnegative
```

```
If Num < 0 Then
```

```
MsgBox “Você deve inserir um número positivo.”
```

```
Exit Sub
```

```
End If
```

```
‘ Insert the square root
```

```
ActiveCell.Value = Sqr(Num)
```

```
End Sub
```

186 Parte III: Conceitos de Programação Uma estrutura If-Then verifica o valor contido na variável Num. Se Num for menor que 0, o procedimento exibe uma caixa de mensagem contendo informações que os humanos podem, de fato, entender. O procedimento termina com a declaração Exit Sub, assim o erro em tempo de execução não tem a oportunidade de acontecer.

A macro ainda não é perfeita

Portanto, o procedimento modificado para EnterSquareRoot é perfeito, certo? Não exatamente. Tente entrar com texto ao invés de um valor. Ou clique o botão Cancelar na caixa de entrada. Essas duas ações geram um erro (Tipos incompatíveis). Esse simples e pequeno procedimento ainda precisa de mais código para lidar com os erros.

O código a seguir usa a função IsNumeric para garantir que Num contém um valor numérico. Se o usuário não entrar com um número, o procedimento exibe uma mensagem e depois é interrompido. Observe ainda que a variável Num agora é definida como uma Variant. Se ela fosse definida como Double, o código geraria um erro não tratado se o usuário tivesse entrado com um valor não numérico na caixa de entrada.

Sub EnterSquareRoot3()

Dim Num As Variant

‘ Tela para valor

Num = InputBox(“Insira um valor”)

‘ Certifique-se que Num seja um número

If Not IsNumeric(Num) Then

MsgBox “Você deve inserir um número.”

Exit Sub

End If

‘Certifique-se que um número não é negativo

If Num < 0 Then

MsgBox “Você deve inserir um número positivo.”

Exit Sub

End If

‘ Insira a raiz quadrada

ActiveCell.Value = Sqr(Num)

End Sub

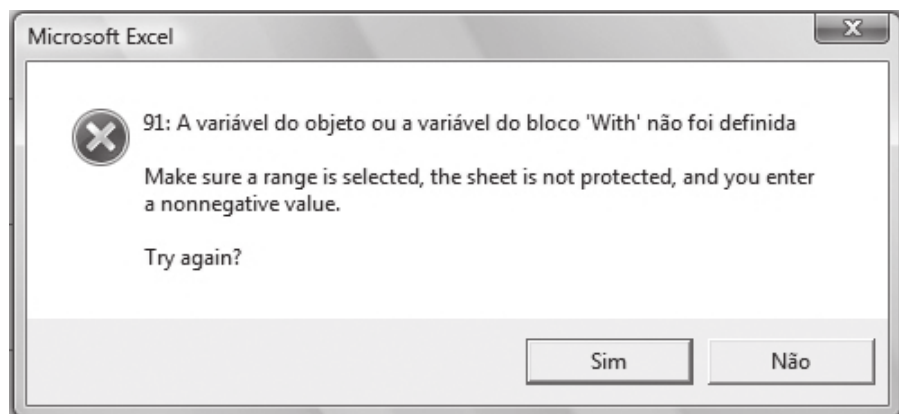
A macro já está perfeita?

Agora, este código está absolutamente perfeito, certo? Ainda não.

Experimente rodar o procedimento enquanto a planilha ativa é uma planilha de gráfico. É, outro erro em tempo de execução, dessa vez é o temido Número 91 (veja a Figura 12-3). Esse erro ocorre porque não há célula ativa quando a planilha de gráficos está ativa, ou quando alguma coisa que não uma faixa é selecionada.

ExcelVBABokks.indb 186

15/02/2013 13:59:49



Capítulo 12: Técnicas de Tratamento de Erros 187

Figura 12-3:

Rodar o

procedimento quando

um gráfico

está selecionado gera

este erro.

A listagem a seguir usa a função TypeName para garantir que a seleção seja uma faixa. Se qualquer outra coisa que não uma faixa for selecionada, este procedimento exibe uma mensagem e depois sai: Sub EnterSquareRoot4()

Dim Num As Variant

‘ Make sure a worksheet is active

If TypeName(Selection) < > “Range” Then

MsgBox “Selecione uma célula para o resultado.”

Exit Sub

End If

‘ Tela para valor

Num = InputBox(“Insira um valor”)

‘ Certifique-se que Num seja uma número

If Not IsNumeric(Num) Then

MsgBox “Você deve inserir um número.”

Exit Sub

End If

‘ Certifique-se que o número não é negativo

If Num < 0 Then

MsgBox “Você deve inserir um número positivo.”

Exit Sub

End If

‘ Insira a raiz quadrada

ActiveVell.Value = Sqr(Num)

End Sub

Desistindo da perfeição

Por ora, este procedimento simplesmente *deve* estar perfeito. Pense de novo, companheiro. Proteja a planilha (use o comando Revisão↔↔

Alterações↔Proteger Planilha) e, depois, rode o código. Sim, uma planilha protegida gera ainda outro erro. E provavelmente, eu não pensei em todos os outros erros que podem acontecer. Continue a ler sobre outra maneira de lidar com erros — até mesmo aqueles que você não pode prever.

ExcelVBABokks.indb 187

15/02/2013 13:59:49

188 Parte III: Conceitos de Programação Como Lidar com Erros de Outra Maneira

Como você pode identificar e lidar com cada erro possível? Com frequência, não pode. Felizmente, o VBA oferece uma outra maneira de lidar com erros.

Reverendo o procedimento EnterSquareRoot

Examine o seguinte código. Eu modifiquei a rotina da seção anterior, acrescentando uma declaração geral On Error para capturar todos os erros e depois verificar se a InputBox foi cancelada.

```
Sub EnterSquareRoot5()
```

```
Dim Num As Variant
```

```
Dim Msg As String
```

```
‘ Configure a manipulação de erros
```

```
On Error GoTo BadEntry
```

```
‘ Tela para valor
```

```
Num = InputBox(“Insira um valor”)
```

```
‘ Sair se cancelado
```

```
If Num = “ “ Then Exit Sub
```

```
‘ Insira a raiz quadrada
```

```
ActiveCell.Value = Sqr(Num)
```

```
Exit Sub
```

```
BadEntry:
```

```
Msg = “ocorreu um erro.” & vbNewLine & vbNewLine
```

```
Msg = Msg & “Assegure-se de ter selecionado um  
intervalo,”
```

```
Msg = Msg & “que a planilha não esteja protegida, “
```

```
Msg = Msg & “e que você não inseriu um valor  
negativo.”
```

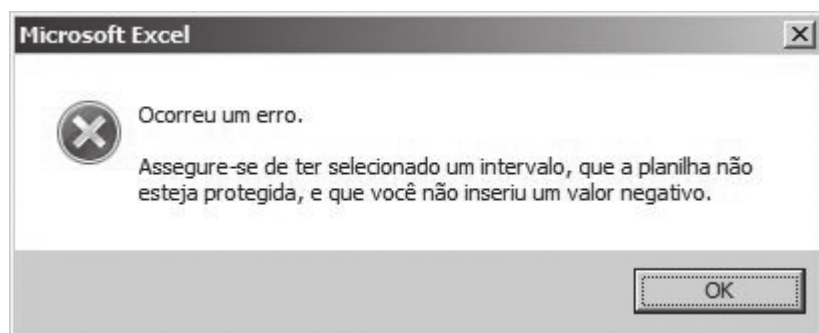
```
MsgBox Msg, vbCritical
```

```
End Sub
```

Esta rotina captura *qualquer* tipo de erro em tempo de execução. Depois de capturar o erro, o procedimento revisado EnterSquareRoot exibe a caixa de mensagem mostrada na Figura 12-4. Esta caixa de mensagem descreve as causas mais prováveis do erro.

ExcelVBABokks.indb 188

15/02/2013 13:59:49



Capítulo 12: Técnicas de Tratamento de Erros 189

DICA

On Error não está funcionando?

Se uma declaração On Error não estiver fun—

4. Assegure-se de que a configuração In—
cionando conforme anunciado, você precisa
terromper em todos os erros não está
trocar uma VBE de suas configurações:
selecionada.

1. Ative o VBE.

Se essa configuração estiver selecionada,

2. Escolha o comando Ferramentas

efetivamente o Excel ignora quaisquer de-

⇒ Opções.

clarações On Error. Em geral, você quer

3. Selecione a guia Geral da caixa de diá-

manter as opções de On Error configurada

logo Opções.

para interromper em erros não tratados.

Figura 12-4:

Um erro em

tempo de

execução

no procedimento gera

esta útil

mensagem

de erro.

Sobre a declaração On Error

Usar uma declaração On Error em seu código VBA permite que você ignore o manuseio de erro integrado do Excel e use o seu próprio código de lidar com erro. No exemplo anterior, um erro em tempo de execução leva a execução da macro a saltar para a declaração rotulada como BadEntry. Como resultado, você evita mensagens de erro pouco amistosas do Excel e pode exibir a sua própria mensagem (mais amistosa, espero) ao usuário.

DICA

Observe que o exemplo usa uma declaração Exit Sub bem antes do rótulo de BadEntry. Essa declaração é necessária pois você não quer

executar o código que lida com erro caso um erro não ocorra.

ExcelVBABokks.indb 189

15/02/2013 13:59:49

190 Parte III: Conceitos de Programação Como Lidar com Erros: Os Detalhes

Você pode usar uma declaração On Error de três formas, conforme mostrado na Tabela 12-1.

Tabela 12-1

Usando a Declaração On Error

Sintaxe

O que ela faz

Rótulo On Error GoTo

Depois de executar essa declaração, o VBA retoma a

execução na linha especificada. Você deve incluir

dois pontos depois do rótulo, para que VBA o

reconheça como um rótulo.

On Error Resume Next

Depois de executar essa declaração, o VBA

simplesmente ignora todos os erros e retoma a

execução com a nova declaração.

On Error GoTo 0

Depois de executar essa declaração, o VBA retoma o

seu comportamento normal de verificação de erro.

Use essa declaração depois de usar uma das outras

declarações de On Error, ou quando quiser remover a

manipulação de erro de seu procedimento.

Recuperação depois de um erro

Em alguns casos, você quer apenas que a rotina termine graciosamente quando ocorre um erro. Por exemplo, você pode exibir uma mensagem descrevendo o erro e depois sair do procedimento (o exemplo EnterSquareRoot5 usa essa técnica). Em outros casos, você quer se recuperar do erro, se possível.

Para se recuperar de um erro, você deve usar uma declaração Resume.

Isso limpa a condição de erro e permite que você prossiga com a execução em algum lugar. É possível usar a declaração Resume de três formas, como mostrado na Tabela 12-2.

Tabela 12-2

Usando a Declaração Resume

Sintaxe

O que ela faz

Resume

A execução retorna com a declaração que causou o erro.

Use isso se o seu código que lida com erro corrigir o

problema e estiver bem para continuar.

Resume Next

A execução retorna com a declaração imediatamente depois

da declaração que causou o erro. Essa ignora,

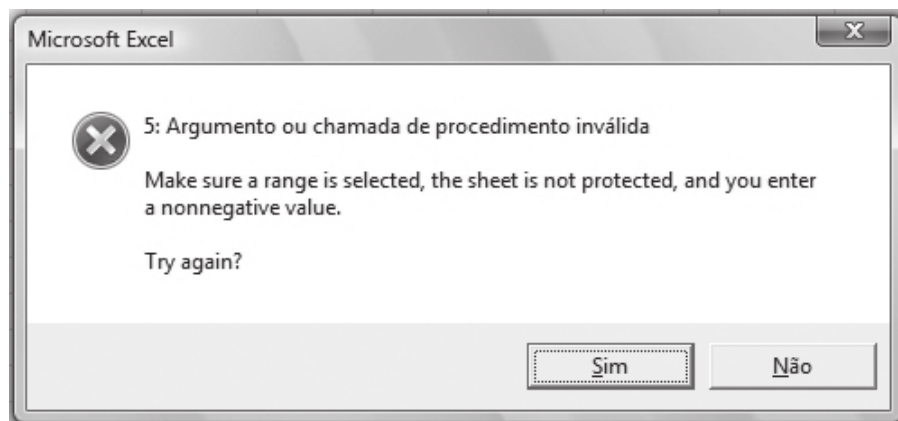
essencialmente, o erro.

Etiqueta Resume

A execução retorna no rótulo que você especifica.

ExcelVBABokks.indb 190

15/02/2013 13:59:49



Capítulo 12: Técnicas de Tratamento de Erros 191

O seguinte exemplo usa uma declaração Resume depois da ocorrência de um erro:

Sub EnterSquareRoot6()

Dim Num As Variant

Dim Msg As String

Dim Ans As Integer

TryAgain:

‘ Configure a manipulação de erros

On Error GoTo BadEntry

‘ Tela para valor

Num = InputBox(“Insira um valor”)

‘ Insira a raiz quadrada

ActiveCell.Value = Sqr(Num)

Exit Sub

BadEntry:

Msg = Err.Number & “ : “ & Error(Err.Number)

Msg = Msg & vbNewLine & vbNewLine

Msg = Msg & “Make sure a range is selected, “

Msg = Msg & “the sheet is not protected, “

Msg = Msg & “and you enter a nonnegative value.”

Msg = Msg & vbNewLine & vbNewLine & “Try again?”

Ans = MsgBox(Msg, vbYesNo + vbCritical)

If Ans = vbYes Then Resume TryAgain

End Sub

Este procedimento tem outro rótulo: TryAgain. Se ocorrer um erro, a execução prossegue na etiqueta BadEntry e o código exibe a mensagem mostrada na Figura 12-5. Se o usuário responder clicando Sim, a declaração Resume pula e a execução volta para a etiqueta TryAgain. Se o usuário clicar Não, o procedimento termina.

Observe que a mensagem de erro também inclui o número do erro, junto com a descrição “oficial” do erro. Tirei isso, porque escrevo a respeito mais adiante. Veja “Como identificar erros específicos”.

Figura 12-5:

Se ocorrer
um erro, o
usuário
pode
decidir se
quer tentar
de novo.

ExcelVBABokks.indb 191

15/02/2013 13:59:50

192 Parte III: Conceitos de Programação Lembre-se que a declaração Resume limpa a condição de erro antes de continuar. Para ver o que quero dizer, experimente substituir a seguinte declaração pela penúltima declaração do exemplo anterior: If Ans = vbYes Then GoTo TryAgain

O código não funciona corretamente se você usar GoTo ao invés de Resume. Para demonstrar, entre com um número negativo: você recebe uma tela de erro. Clique Sim para tentar de novo e depois, entre com *um outro* número negativo. Esse segundo erro não é capturado, pois a condição original de erro não foi removida.

Este exemplo está disponível no site deste livro.

Lidando com erros resumidamente

Para ajudá-lo a manter claro esse negócio de lidar com erro, eu preparei um resumo curto. Um bloco de código para lidar com erros tem as seguintes características:

5 Ele começa logo depois do rótulo especificado na declaração On

Error.

5 Ele só deveria ser atingido pela sua macro se ocorrer um erro.

Isso significa que você deve usar uma declaração, tal como Exit Sub ou ExitFunction, imediatamente antes do rótulo.

5 Ele pode exigir uma declaração Resume. Se você decidir não abortar o procedimento quando ocorrer um erro, deve executar uma declaração Resume antes de voltar ao código principal.

Como saber quando ignorar erros

Em alguns casos, está perfeitamente certo ignorar erros. É quando a declara-

ção On Error Resume Next entra em cena.

O seguinte exemplo faz uma loop através de cada célula na faixa selecionada e converte o valor para a sua raiz quadrada. Este procedimento gera uma mensagem de erro se qualquer célula na seleção contiver um número negativo ou texto: Sub SelectionSqrt()

Dim cell As Range

If TypeName(Selection) <> "Range" Then Exit Sub

For Each cell In Selection

cell.Value = Sqr(cell.Value)

Next cell

End Sub

ExcelVBABokks.indb 192

15/02/2013 13:59:50

Capítulo 12: Técnicas de Tratamento de Erros 193

Neste caso, você pode querer simplesmente pular qualquer célula que contenha um valor que não pode ser convertido a uma raiz quadrada.

Você pode criar todos os tipos de capacidades de verificação de erro, usando estruturas If-Then, mas pode conceber uma solução melhor (e mais simples), apenas ignorando os erros que acontecem.

A seguinte rotina consegue isso, usando a declaração On Error Resume Next: Sub SelectionSqrt()

```
Dim cell As Range
```

```
If TypeName(Selection) <> "Range" Then Exit Sub
```

```
On Error Resume Next
```

```
For Each cell In Selection
```

```
cell.Value = Sqr(cell.Value)
```

```
Next cell
```

```
End Sub
```

Em geral, você pode usar uma declaração On Error Resume Next se considerar os erros inconsequentes à sua tarefa.

Como identificar erros específicos

Os erros não são todos criados iguais. Alguns são sérios e outros nem tanto. Embora você possa ignorar erros que considera sem consequência, você deve lidar com outros erros, mais sérios. Em alguns casos, é preciso identificar o erro específico que ocorreu.

Na ocorrência de um erro, o Excel armazena o número do erro em um objeto Error, chamado Err. A propriedade Number desse objeto contém o número do erro. Você recebe uma descrição do erro usando a função VBA Error. Por exemplo, a seguinte declaração exibe o número do erro e uma descrição: MsgBox Err.Number & ": " & Error(Err.Description)

A Figura 12-5, apresentada anteriormente, mostra um exemplo disso.

Porém, tenha em mente que as mensagens de erro do Excel nem sempre são úteis — mas, você já sabe disso.

O procedimento a seguir demonstra como determinar qual erro aconteceu. Nesse caso, você pode ignorar com segurança erros causados por tentar obter a raiz quadrada de um número não positivo (isto é, o erro 5), ou erros causados por tentar obter a raiz quadrada de um valor não numérico (erro 13). Por outro lado, você precisa informar ao usuário se a planilha está protegida e a seleção contém uma ou mais células bloqueadas (caso contrário, o usuário pode pensar que a macro funcionou, quando na verdade não o fez). Esse evento causa o erro 1004.

ExcelVBABokks.indb 193

15/02/2013 13:59:50

194 Parte III: Conceitos de Programação Sub SelectionSqrt()

```
Dim cell As Range
```

```
Dim ErrMsg As String
```

```
If TypeName(Selection) < > "Range" Then Exit Sub
```

```
On Error GoTo ErrorHandler
```

```
For Each cell In Selection
```

```
cell.Value = Sqr(cell.Value)
```

```
Next cell
```

Exit Sub

ErrorHandler:

Select Case Err.Number

Case 5 'Negative number

Resume Next

Case 13 'Type mismatch

Resume Next

Case 1004 'Locked cell, protected sheet

MsgBox "Cell is locked. Try again.",

vbCritical,

cell.Address

Exit Sub

Case Else

ErrMsg = Error(Err.Number)

MsgBox "ERROR: " & ErrMag, vbCritical, cell,

Address

Exit Sub

End Select

End Sub

Quando ocorre um erro em tempo de execução, a execução pula para o código iniciando no rótulo ErrorHandler. A estrutura Select Case (eu discuto esta estrutura no Capítulo 10) testa três números comuns de erros. Se o número do erro for 5 ou 13, a execução retorna na declaração seguinte (em outras palavras, o erro é ignorado). Porém, se o número do erro for 1004, a rotina informa ao usuário e depois termina. O último caso, um punhado de erros não previstos captura todos os outros erros e exibe a mensagem de erro atual.

Um Erro Intencional

Às vezes, você pode usar um erro em seu benefício. Por exemplo, suponha que você tem uma macro que só funciona se determinada pasta de trabalho estiver aberta. Como você pode determinar se aquela pasta de trabalho está aberta? Uma maneira é escrever código que faça loops através da coleção Workbooks verificando, para determinar se a pasta de trabalho na qual está interessado está naquela coleção.

Capítulo 12: Técnicas de Tratamento de Erros 195

Aqui está um modo mais fácil: uma função mais geral aceita um argumento (um nome de workbook) e retorna True se o workbook estiver aberto, False se não estiver.

Eis a função:

```
Function WorkbookIsOpen(book As String) As Boolean
```

```
Dim WBName As String
```

```
On Error GoTo NotOpen
```

```
WBName = Workbooks(book).Name
```

```
WorkbookIsOpen = True
```

```
Exit Function
```

```
NotOpen:
```

```
WorkbookIsOpen = False
```

```
End Function
```

Esta função tem a vantagem de que o Excel gera um erro se você fizer referência a uma pasta de trabalho que não está aberta. Por exemplo, a declaração a seguir gera um erro se uma pasta de trabalho chamada MyBook.xlsl não estiver aberta: WBName = Workbooks("MyBook.xlsl").Name

Na função WorksheetsOpen, a declaração On Error diz ao VBA para retornar a macro à declaração NotOpen se houver um erro. Portanto, um erro significa que a pasta de trabalho não está aberta, e a função retorna False.

Se a pasta de trabalho estiver aberta, não há erro e a função retorna True.

Eis outra variação da função WorksheetsOpen. Esta versão usa On Error Resume Next para ignorar o erro. Porém, o código verifica a propriedade Number de Err. Se Err.Number for 9, nenhum erro ocorreu e a pasta de trabalho está aberta. Se Err.Number for qualquer outra coisa, significa que ocorreu um erro (e a pasta de trabalho não está aberta).

```
Function WorkbookIsOpen(book) As Boolean
```

```
Dim WBName As String
```

```
On Error Resume Next
```

```
WBName = Worksheets(book).Name
```

```
If Err.Number = 0 Then WorkbookIsOpen = True _
```

```
Else WorkbookIsOpen = False
```

```
End Function
```

O exemplo a seguir demonstra como usar esta função em um procedimento Sub:

```
Sub UpdatePrices()
```

```
If Not WorkbookIsOpen("Prices.xls") Then
```

MsgBox “Por favor abra a pasta de trabalho

Prices primeiro!”

Exit Sub

End If

‘ [Outros códigos entram aqui]

End Sub

ExcelVBABokks.indb 195

15/02/2013 13:59:50

196 Parte III: Conceitos de Programação O procedimento UpdatePrices (o qual deve estar na mesma pasta de trabalho que WorkbookIsOpen) chama a função WorkbookIsOpen e passa o nome da pasta de trabalho (Prices.xlsl) como um argumento. A função WorkbookIsOpen retorna Verdadeiro ou Falso. Portanto, se a pasta de trabalho não estiver aberta, o procedimento informa o fato ao usuário. Se a pasta de trabalho estiver aberta, a macro prossegue.

Lidar com erro pode ser uma proposta ardilosa — afinal, pode ocorrer muitos erros diferentes e você não pode prever todos eles.

Normalmente, se possível, você deveria capturar erros e corrigir a situação antes que o Excel interfira. Escrever código eficiente de captura de erro requer um conhecimento pormenorizado de Excel e um claro entendimento de como funciona o tratamento de erros do VBA. Capítulos posteriores contêm mais exemplos de como lidar com erros.

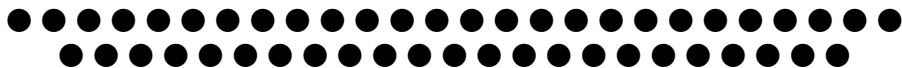
ExcelVBABokks.indb 196

15/02/2013 13:59:50

Capítulo 13

Técnicas de Extermínio

de Bugs



Neste Capítulo

- ☐ Definição de bug e porque você deveria esmagá-lo
- ☐ Como reconhecer tipos de bugs de programa que você pode encontrar
- ☐ Usando técnicas para depurar o seu código
- ☐ Como usar ferramentas de depuração integradas ao VBA



Se a palavra *bug* evoca a imagem de um coelho de desenho animado, este capítulo pode deixá-lo determinado. Colocando de forma simples, um bug é um erro em sua programação. Aqui, eu abordo o tópico de bugs de programação — como identificá-los e como tirá-los da face de seu módulo.

Espécies de Bugs

Bem-vindo à Entomologia 101. O termo bug de programa, como provavelmente você sabe, refere-se a um problema com software. Em outras palavras, se o software não executar conforme esperado, ele tem um bug. A verdade é que todos os principais programas de software têm bugs muitos bugs. O próprio Excel tem centenas (se não milhares) de bugs. Felizmente, a grande maioria desses bugs é relativamente obscura e só aparece em circunstância bem específicas.

Quando você escreve programas VBA pouco comuns, provavelmente o seu código terá bugs. Isso é um fato e não necessariamente um reflexo de sua habilidade de programação. Os bugs podem estar em qualquer das seguintes categorias:

5 Falhas lógicas em seu código: Com frequência, você pode evitar esses bugs, pensando cuidadosamente no problema que o seu programa apresenta.

5 Bugs no contexto incorreto: Esse tipo de bug surge quando você tenta fazer algo na hora errada. Por exemplo, você pode tentar

escrever dados em células na planilha ativa quando, na verdade, ela é uma planilha de gráfico (que não tem células).

5 Bugs de casos extremos: Esses bugs levantam suas caras feias quando o programa encontra dados não previstos, tais como

números muito grandes ou muito pequenos.

ExcelVBABokks.indb 197

15/02/2013 13:59:50

198 Parte III: Conceitos de Programação 5 Bugs de tipo errado de dados: Esse tipo de bug acontece quando você tenta processar dados do tipo errado, tal como tentar tirar a raiz quadrada de uma string de texto.

5 Bugs de versão errada: Esse tipo de bug envolve

incompatibilidades entre diferentes versões do Excel. Por

exemplo, você pode desenvolver uma pasta de trabalho com

Excel 2010 e, depois, descobrir que a pasta de trabalho não

funciona com o Excel 2003. Normalmente é possível evitar tais problemas evitando usar os recursos específicos de versão. Com frequência, a abordagem mais fácil é desenvolver o seu aplicativo usando a versão de número mais baixo do Excel que os usuários podem ter. No entanto, em todos os casos, você deve testar o seu trabalho em todas as versões que imagina que serão usadas.

5 Bugs além do seu controle: Esses são os mais frustrantes. Um exemplo acontece quando a Microsoft atualiza o Excel e faz uma alteração menor, não documentada, que leva a sua macro a estourar. Até atualizações de segurança têm sido conhecidas

por causar problemas.

Depurar (debugging) é o processo de identificar e corrigir bugs em seu programa. Leva tempo para desenvolver habilidades de depuração,

assim, não fique desencorajado se esse processo for difícil no início.

É importante entender a distinção entre *bugs* e *erros de sintaxe*. Um erro **BRE-SE**

de sintaxe é um erro de linguagem. Por exemplo, você poderia soletrar **LEM**

errado uma palavra-chave, omitir a declaração Next (Próxima) em um loop For-Next (Para a Próxima), ou ter um parêntese incompatível.

Antes de poder sequer executar o procedimento, você deve corrigir esses erros de sintaxe. Um bug de programa é muito mais sutil. É

possível executar a rotina, mas ela não executa conforme esperado.

Como Identificar Bugs

Antes de fazer qualquer depuração, você deve determinar se, de fato, existe um bug. Você pode dizer que a sua macro contém um bug se ela não funcionar da maneira como deveria. Puxa, este livro está cheio de perspicácias, não é? Geralmente, mas nem sempre você pode perceber isso com facilidade.

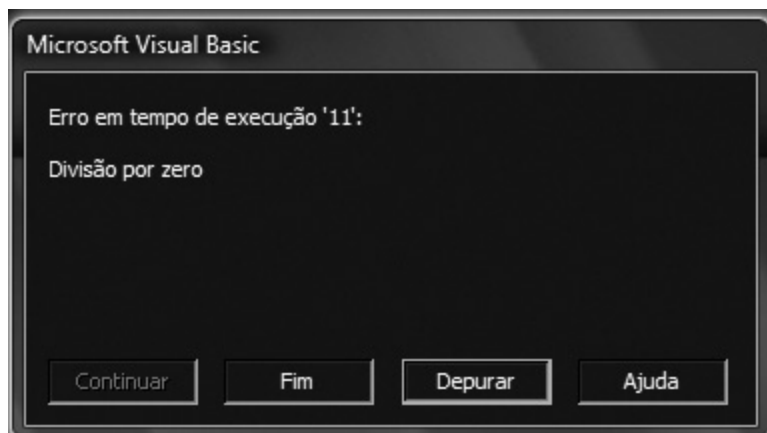
Com frequência (mas nem sempre), um bug fica claro quando o Excel exibe uma mensagem de erro em tempo de execução. A Figura 13-1 mostra um exemplo. Observe que essa mensagem de erro inclui um botão chamado Depurar. Mais sobre isso, mais adiante, na seção “Sobre o Depurador”.

Um fato importante conhecido por todos os programadores é que, frequentemente, bugs aparecem quando você menos espera. Por

exemplo, só porque a sua macro funciona bem com um conjunto de dados, não significa que você pode supor que ela funcionará bem da mesma forma com todos os conjuntos de dados.

ExcelVBABokks.indb 198

15/02/2013 13:59:50



Capítulo 13: Técnicas de Extermínio de Bugs 199

Figura 13-1:

Uma
mensagem
de erro como
esta
frequentemente significa que o seu
código VBA
contém um
bug.

A melhor abordagem de depuração é testar cuidadosamente, sob uma variedade de condições de vida real. E porque quaisquer alterações em sua pasta de trabalho feitas pelo seu código VBA não podem ser desfeitas, é sempre uma boa ideia usar uma cópia de segurança da pasta de trabalho que usar para testar. Normalmente, eu copio alguns arquivos em uma pasta temporária e uso tais arquivos para o meu teste.

Técnicas de Depuração

Nesta seção, discuto os quatro métodos mais comuns para depurar

5 Examinar o código

5 Inserir funções MsgBox em vários locais em seu código

5 Inserir declarações Debug.Print

5 Inserir as ferramentas de depuração integradas no Excel

Como examinar o seu código

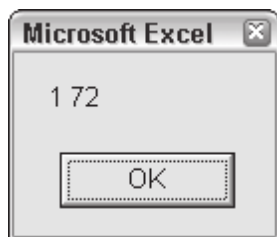
Talvez a técnica de depuração mais direta seja simplesmente dar uma boa olhada em seu código, para ver se é possível encontrar o problema.

Se você tiver sorte, o erro logo aparece e você bate na testa e diz ,
“Oh!”

Quando a dor na testa diminuir, você pode corrigir o problema.

Veja que eu disse “Se você tiver sorte”. Isso porque, com frequência, você descobre erros quando esteve trabalhando em seu programa por oito horas direto, são duas horas da madrugada e você está funcionando à base de cafeína e força de vontade. Em ocasiões assim, você tem sorte se puder ao menos ver seu código, ficando sozinho para encontrar os bugs.

Assim, não se surpreenda se, simplesmente examinar seu código não for o bastante para fazê-lo encontrar e eliminar todos os bugs que ele contém.



200 Parte III: Conceitos de Programação

Usando a função MsgBox

Um problema comum em muitos programas envolve uma ou mais

variáveis que não tomam os valores que você espera. Em tais casos, monitorar a(s) variável(eis) enquanto o seu código roda é uma técnica útil de depuração. Uma maneira de fazer isso é inserir temporariamente funções MsgBox em sua rotina. Por exemplo, se você tiver uma variável chamada CellCount, pode inserir a seguinte declaração: MsgBox CellCount

Quando você executa a rotina, a função MsgBox exibe o valor de CellCount.

Em geral, é útil exibir os valores de duas ou mais variáveis na caixa de mensagem. A seguinte declaração exibe o valor atual de duas variáveis: LoopIndex(1) e CellCount(72), conforme mostrado na Figura 13-2: MsgBox LoopIndex & " " & CellCount **Figura 13-2:**

Usando uma

caixa de

mensagem

para exibir o

valor de

duas

variáveis.

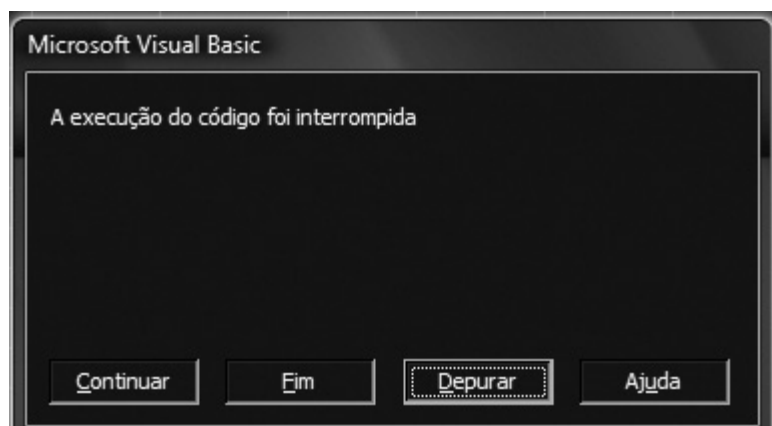
Observe que eu combino as duas variáveis com o operador de concatenação (&) e insiro um caractere de espaço entre elas. Caso contrário, a caixa de mensagem conecta os dois valores, fazendo-os parecer como um único valor. Também é possível usar a constante

integrada `vbNewLine`, no lugar do caractere de espaço. `vbNewLine` insere uma linha de alimentação de pausa, a qual exibe o texto em uma nova linha. A seguinte declaração exibe três variáveis, cada uma em uma linha separada: `MsgBox LoopIndex & vbNewLine & CellCount & _`

`vbNewLine & MyVal`

Esta técnica não é limitada a monitorar variáveis. Você pode usar uma caixa de mensagem para exibir todo o tipo de informações úteis enquanto o seu código estiver rodando. Por exemplo, se o seu código fizer loops através de uma série de planilhas, a seguinte declaração exibe o nome e o tipo da planilha ativa: `MsgBox ActiveSheet.Name & “ “ & TypeName(ActiveSheet) ExcelVBABokks.indb 200`

15/02/2013 13:59:51



Capítulo 13: Técnicas de Extermínio de Bugs 201

Se a sua caixa de mensagem exibir algo inesperado, pressione

`Ctrl + Break` e será exibida uma caixa de diálogo que informa, “A Execu-

ção do código foi interrompida”. Como mostrado na Figura 13-3, você tem quatro escolhas:

5 Clicar no botão Continuar e o código continuará a executar

5 Clicar no botão Fim e a execução termina.

5 Clicar no botão Depurar e o VBE vai para o modo de depuração (o qual é explicado um pouco mais adiante).

5 Clicar no botão Ajuda e uma tela de ajuda informa que você pressionou Ctrl + Break. Em outras palavras, não é muito útil.

Figura 13-3:

Pressionar

Ctrl + Break

interrompe a

execução de

seu código e

lhe dá

algumas

escolhas.

Fique à vontade para usar as funções MsgBox frequentemente ao **BRE-SE**

depurar o seu código. Assegure-se apenas de removê-las depois de **LEM**

identificar e corrigir o problema.

Inserindo declarações Debug.Print

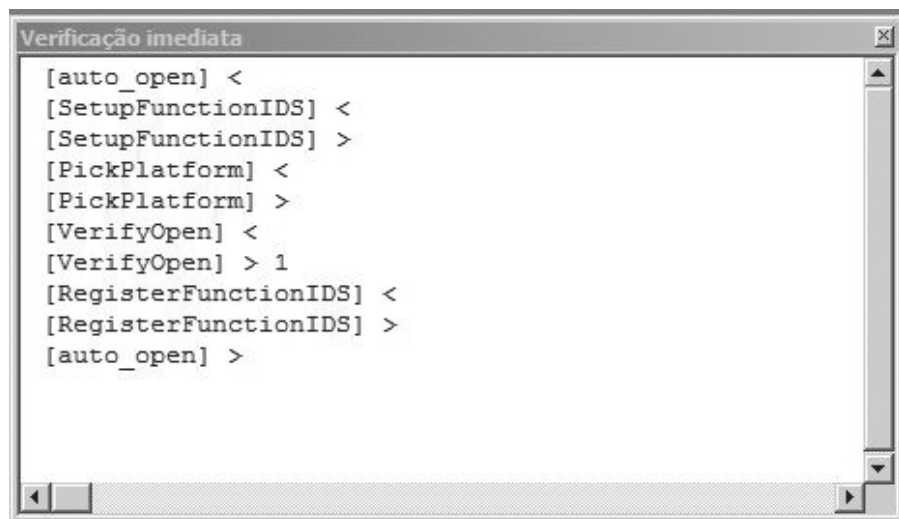
Como uma alternativa ao uso de MsgBox em seu código, você pode inserir uma ou mais declarações temporárias Debug.Print. Use essas declarações para imprimir o valor de uma ou mais variáveis na janela Verificação imediata. Eis um exemplo que exibe o valor de três variáveis: Debug.Print LoopIndex, CellCount, MyVal Observe que as

variáveis são separadas por vírgulas. Você pode exibir tantas variáveis quanto quiser com uma única declaração `Debug.Print`. Se a janela Verificação imediata do VBE não estiver visível, pressione `Ctrl + G`.

Diferente de `MsgBox`, as declarações `Debug.Print` não interrompem o seu código. Assim, você precisará ficar atento à janela Verificação imediata para ver o que está acontecendo.

Depois de ter depurado seu código, assegure-se de remover todas as declarações `Debug.Print`. Mesmo grandes empresas, como a Microsoft, `ExcelVBABokks.indb 201`

15/02/2013 13:59:51



202 Parte III: Conceitos de Programação

ocasionalmente se esquecem de remover suas declarações `Debug.Print`.

Em diversas versões anteriores do Excel, cada vez que o add-in Analysis ToolPak (Pacote de Ferramentas de Análise) era aberto, você via diversas mensagens estranhas na janela Verificação imediata (como mostrado na Figura 13-4). O problema foi corrigido no Excel 2007.

Figura 13-4:

Até programadores

profissionais

às vezes se
esquecem de
remover suas
declarações

Debug.Print.

Usando o depurador VBA

Os projetistas de Excel são intimamente familiarizados com o conceito de bugs, portanto, o Excel inclui um conjunto de ferramentas de depuração que pode ajudá-lo a corrigir problemas em seu código VBA.

O depurador VBA é o tópico da próxima seção.

Sobre o Depurador

Nesta seção, eu discuto os sangrentos detalhes de usar as ferramentas de depuração do Excel. Essas ferramentas são muito mais poderosas do que as técnicas discutidas na seção anterior. Mas, junto com o poder vem a responsabilidade. Usar as ferramentas de depuração requer um pouco de trabalho de configuração.

Configurando pontos de

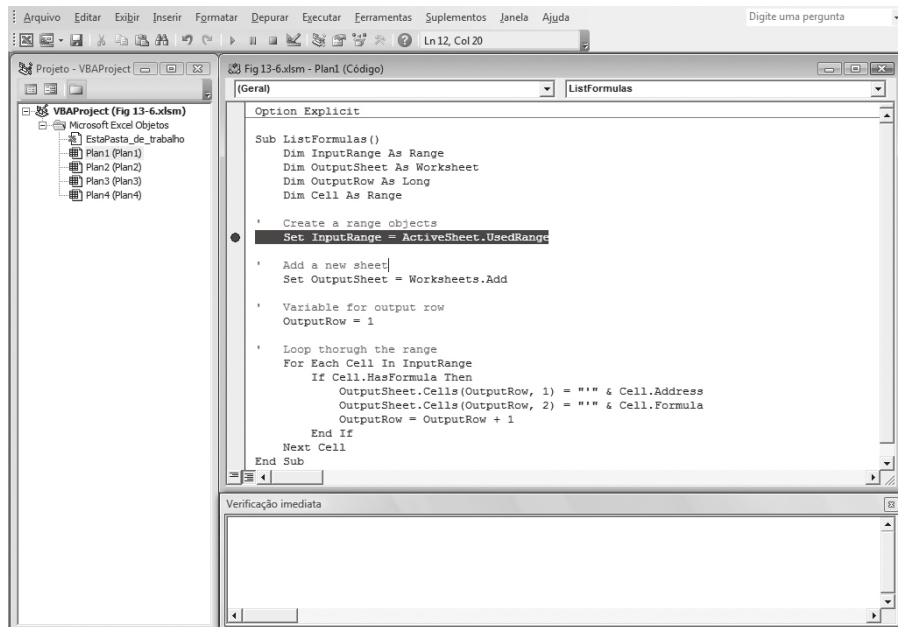
interrupção em seu código

Anteriormente neste capítulo, eu defendi o uso de funções MsgBox em seu código para monitorar os valores de determinadas variáveis. Exibir uma caixa de mensagem interrompe, essencialmente, o seu código no meio da execução e clicar o botão OK retorna a execução.

Não seria bom se você pudesse interromper a execução de uma rotina, dar uma olhada no valor de *qualquer* de suas variáveis e depois continuar a execução? Bem, isso é exatamente o que você pode fazer configurando um ponto de interrupção. Você pode configurar um ponto de interrupção em seu código VBA de várias maneiras:

ExcelVBABokks.indb 202

15/02/2013 13:59:51



Capítulo 13: Técnicas de Extermínio de Bugs 203

5 Mova o cursor para a declaração em que deseja que a execução seja interrompida; depois, pressione F9.

5 Clique na margem cinza, à esquerda da declaração onde você deseja que a execução pare.

5 Posicione o ponto de inserção na declaração onde deseja que a execução pare. Depois, use o comando Depurar → Ativar Ponto de Interrupção.

5 Clique com o botão direito uma declaração e escolha Ativar → Ponto de Interrupção a partir do menu de atalho.

Os resultados que se obtém ao configurar um ponto de interrupção são apresentados na Figura 13-5. O Excel destaca a linha para lembrá-lo que você configura um ponto de interrupção lá; ele também insere um ponto grande na margem cinza.

Figura 13-5:

A declaração

destacada

marca um

ponto de

interrupção

neste

procedimento.

Quando você executa um procedimento, o Excel entra no Modo de interrupção antes da linha com o ponto de interrupção ser executado.

No modo de interrupção, a palavra [interromper] é exibida na barra de título do VBE. Para sair do modo de interrupção e continuar a execu-

ção, pressione F5 ou clique o botão Executar Sub/UserForm na barra de ferramentas do VBE. Veja “Percorrendo o seu código”, mais adiante neste capítulo, para saber mais.

Para remover rapidamente um ponto de interrupção, clique o ponto

DICA

grande na margem cinza e mova o cursor para a linha destacada e pressione F9. Para remover todos os pontos de interrupção no módulo, pressione Ctrl + Shift + F9.

ExcelVBABokks.indb 203

15/02/2013 13:59:51

204 Parte III: Conceitos de Programação O VBA também tem uma palavra-chave que força o modo de interrupção: Stop

Quando o seu código atingir a palavra-chave Stop, o VBA entra no

modo de interrupção. A coisa jeitosa sobre essa palavra Stop é que, se o seu código for protegido, ele será ignorado.

O que é modo de interrupção? Você pode pensar nele como uma posição de animação suspensa. O seu código VBA para de rodar e a declaração atual é destacada em amarelo brilhante. No modo de interrupção você pode: 5 Digitar declarações VBA na janela Verificação imediata. Para detalhes, veja a próxima seção.

5 Pressionar F8 para caminhar pelo seu código uma linha de cada vez, para verificar diversas coisas enquanto o programa está parado.

5 Mover o cursor do mouse sobre uma variável para exibir o seu valor em uma pequena janela pop-up.

5 Pular a(s) próxima(s) declaração(ões) e continuar a execução lá (ou mesmo, voltar um par de declarações).

5 Editar uma declaração e depois continuar.

A Figura 13-6 mostra alguma depuração em ação. Um ponto de interrupção-DICA

ção é ajustado (observe o grande ponto) e eu usei a tecla F8 para percorrer o código, linha por linha (veja a seta que indica para a declaração atual). Eu usei a janela Verificação imediata para observar algumas coisas, o cursor do mouse está pairando sobre a variável OutputRow e o VBE exibe o seu valor atual.

Usando a janela Verificação imediata

A janela Immediate não pode ser visível no VBE. Você pode exibir a janela Immediate (Imediata) do VBE em qualquer ocasião, pressionando Ctrl + G.

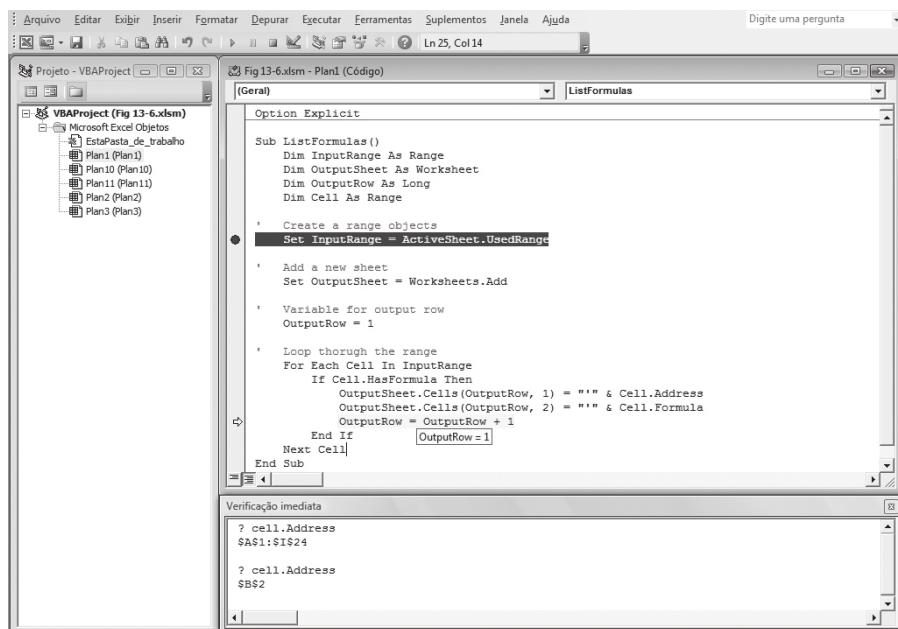
No modo Break, a janela Immediate é especialmente útil para encontrar o valor atual de qualquer variável em seu programa. Por exemplo, se você quiser saber o valor atual de uma variável chamada CellCount, entre com o seguinte na janela Immediate e pressione Enter

(Entrar): Print CellCount Você pode poupar alguns milésimos de segundos usando um ponto de interrogação no lugar da palavra *Print* (Imprimir), assim:

? CellCount

ExcelVBABokks.indb 204

15/02/2013 13:59:52



Capítulo 13: Técnicas de Extermínio de Bugs 205

Figura 13-6:

Um cenário

típico no

modo de

interrupção.

A janela Verificação imediata permite que você faça outras coisas além de verificar valores de variável. Por exemplo, é possível mudar o valor de uma variável, ativar uma planilha diferente ou mesmo abrir

uma nova pasta de trabalho. Apenas, assegure-se de que o comando que você inserir seja uma declaração VBA válida.

Você também pode usar a janela Verificação imediata quando o Excel **DICA**

não estiver no modo de interrupção. Frequentemente, eu uso a janela Verificação imediata para testar pequenos fragmentos de código (sempre que posso comprimir em uma única linha) antes de incorporá-los em meus procedimentos.

Percorrendo o seu código

Enquanto no modo de interrupção, você também pode percorrer o seu código, linha por linha. Uma declaração é executada cada vez que você pressionar F8. Através dessa execução linha por linha de seu código, você pode ativar a janela Verificação imediata em qualquer ocasião, para verificar a posição de suas variáveis.

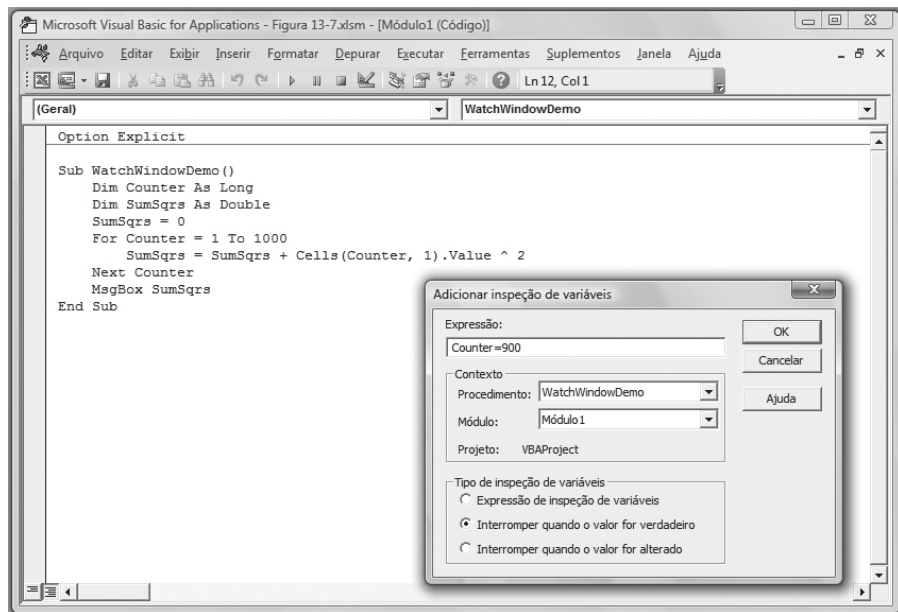
Você pode usar o seu mouse para alterar qual declaração o VBA **DICA**

executará em seguida. Se colocar o cursor do seu mouse na margem cinza à esquerda da declaração atualmente destacada (que normalmente será amarela), o seu cursor muda para uma seta indicando à direita.

Simplesmente arraste o seu mouse para a declaração a ser executada em seguida e veja que a declaração fica amarela.

ExcelVBABokks.indb 205

15/02/2013 13:59:52



206 Parte III: Conceitos de Programação

Usando a janela Inspeção de Variáveis

Em alguns casos, você pode querer saber se determinada variável ou expressão toma um valor em especial. Por exemplo, suponha que um procedimento faz loops através de 1.000 células. Você percebe que houve um problema durante a 900ª iteração do loop. Bem, você poderia inserir um ponto de interrupção no loop, mas isso significaria responder a 899 solicitações antes de o código finalmente obter a iteração que você quer ver (e isso fica chato bem depressa). Uma solução mais eficaz envolve configurar inspeção de variáveis.

Por exemplo, você pode usar esse recurso colocando o procedimento no modo de interrupção sempre que determinada variável tomar um valor específico — por exemplo, Counter = 900. Para criar uma expressão de inspeção, escolha Depurar > Adicionar inspeção de variáveis para exibir a caixa respectiva de diálogo. Veja a Figura 13-7.

Figura 13-7:

A caixa de

diálogo

Adicionar

inspeção de
variáveis
permite que
você especifique uma
condição que
causa uma
interrupção.

A caixa de diálogo Adicionar inspeção de variáveis tem três partes: 5
Expressão: Entre com uma expressão VBA válida ou variável aqui.
Por exemplo, *Counter = 900* ou apenas *Counter*.

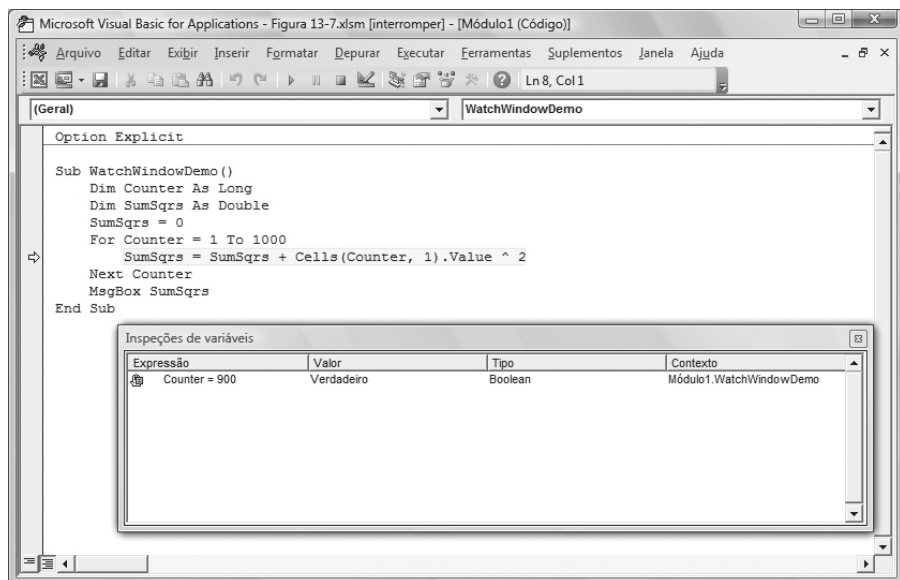
5 **Contexto:** Selecione o procedimento e o módulo que deseja observar. Veja que é possível selecionar Todos os Procedimentos e Todos os Módulos.

5 **Tipo de inspeção de variáveis:** Selecione o tipo de inspeção de variáveis, selecionando uma das opções. A sua escolha aqui

depende da expressão fornecida. A primeira escolha, Expressão de inspeção de variáveis não gera uma interrupção, mas apenas exibe o valor da expressão quando ocorre uma interrupção.

ExcelVBABokks.indb 206

15/02/2013 13:59:52



Capítulo 13: Técnicas de Extermínio de Bugs 207

Execute o seu procedimento depois de configurar a sua inspeção de variável. As coisas rodam normalmente até que a sua expressão seja satisfeita (com base no Tipo de inspeção que você especificou).

Quando isso acontecer, o Excel entra no modo de interrupção (você configurou o Tipo de inspeção para Interromper quando o valor for verdadeiro, não foi?). A partir daí, você pode percorrer o código ou usar a janela Verificação imediata para depurar seu código.

Quando você cria uma inspeção, o VBE exibe a janela Inspeção de variáveis mostrada na Figura 13-8. Essa janela exibe o valor de todas as inspeções que você definiu. Nesta figura, o valor de Counter atinge 900, o que levou o Excel a entrar no modo de interrupção.

Figura

13-8:

A janela

Inspeção

de

variáveis

exibe todas

as

inspeções.

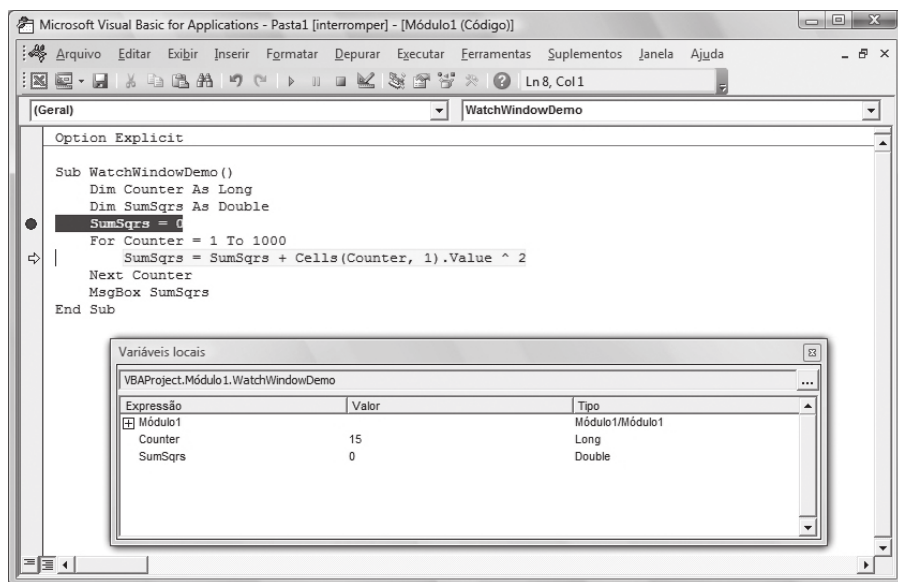
A melhor maneira de entender como funciona esse negócio de inspeção é usá-lo e tentar várias opções. Em pouco tempo, provavelmente você vai pensar em como conseguia viver sem ele.

Usando a janela Variáveis locais

Uma outra ajuda útil de depuração é a janela Variáveis locais. Você pode exibir essa janela escolhendo Exibir>Janela ‘Variáveis locais’ a partir do menu do VBE. Quando você estiver no modo de interrupção, essa janela exibirá uma lista com todas as variáveis que são locais ao procedimento atual (veja a Figura 13-9). Uma coisa boa sobre essa janela é que você não precisa acrescentar manualmente uma carga de inspeção se quiser olhar o conteúdo de muitas variáveis. O VBE fez todo o trabalho duro para você.

ExcelVBABokks.indb 207

15/02/2013 13:59:52



208 Parte III: Conceitos de Programação

Figura 13-9: A

janela

variáveis

locais exibe o

conteúdo das

Variáveis do

procedimento

atual.

Dicas para Redução de Bugs

Eu não posso dizer a você como eliminar bugs completamente de seus programas. Encontrar bugs em software pode ser uma profissão, por si só, mas eu posso oferecer algumas dicas para ajudá-lo a minimizar esses bugs:

5 Use uma declaração Option Explicit no início de seus módulos.

Essa declaração exige que você defina o tipo de dados para cada variável que usar. Isso dá um pouco mais de trabalho, mas você evita o erro comum de digitar incorretamente o nome de uma variável. E isso tem um belo efeito colateral: as suas rotinas rodam um pouco mais depressa.

5 Formate o seu código com recuo. Usar recuos ajuda a delinear diferentes segmentos de código. Se o seu programa tiver vários loops For-Next aninhados, por exemplo, o uso de recuos ajuda a controlar todos eles.

5 Tenha cuidado com a declaração On Error Resume Next.

DICA

Como discuto no Capítulo 12, essa declaração leva o Excel a ignorar quaisquer erros e continua a executar a rotina. Em alguns casos, usar essa declaração leva o Excel a ignorar erros que ele

não deveria ignorar. O seu código pode ter bugs e você pode nem ao menos saber.

5 Use muitos comentários. Nada é mais frustrante do que rever o código que você escreveu seis meses atrás e não ter uma dica

de como ele funciona. Acrescentando alguns comentários para descrever a sua lógica, você pode poupar muito tempo pelo caminho.

ExcelVBABokks.indb 208

15/02/2013 13:59:53

Capítulo 13: Técnicas de Extermínio de Bugs 209

5 Mantenha os seus procedimentos Sub e Function simples.

Escrevendo o seu código em pequenos módulos, cada qual com um único objetivo bem definido, você simplifica o processo de depuração.

5 Use o gravador de macro para ajudar na identificação de

propriedades e métodos. Quando eu não consigo lembrar o nome ou a sintaxe de uma propriedade ou método,

frequentemente eu apenas gravo uma macro e verifico o código gravado.

5 Entenda o depurador do Excel. Ainda que de início possa parecer um pouco desanimador, o depurador do Excel é uma

ferramenta útil. Invista algum tempo para conhecê-lo.

Depurar código não é uma das minhas atividades preferidas (tem uma alta classificação ao ser auditada pela Receita Federal), mas é um mal necessário que acompanha a programação. Quanto mais experiência você adquire com VBA, menos tempo você gasta depurando e, quando tiver que depurar, torna-se mais eficiente em fazê-lo.

ExcelVBABokks.indb 209

15/02/2013 13:59:53

210 Parte III: Conceitos de Programação ExcelVBABokks.indb 210

15/02/2013 13:59:53

Capítulo 14

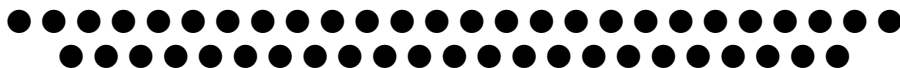
Exemplos de Programação

em VBA



Neste Capítulo

- ☐ Explorando exemplos de VBA
- ☐ Como fazer o seu código VBA rodar o mais rápido possível



A minha filosofia para descobrir como escrever macros em Excel se baseia pesadamente em exemplos. Descobri que um bom

exemplo geralmente comunica um conceito muito melhor do que uma longa descrição da teoria subjacente. Pelo fato de estar lendo este livro, provavelmente você concorda comigo. Este capítulo apresenta vários exemplos que demonstram técnica comuns em VBA.

Esses exemplos são organizados nas seguintes categorias:

5 Como trabalhar com faixas

5 Como alterar configurações do Excel

5 Como trabalhar com gráficos

5 Como dar velocidade ao seu código VBA

Embora você possa ser capaz de usar diretamente alguns desses exemplos, na maioria dos casos você deve adaptá-los às suas próprias necessidades.

Como Trabalhar com Ranges (faixas)

Provavelmente, a maior parte de sua programação em VBA envolve faixas (para recordar a questão sobre objetos Range, consulte o Capítulo 8). Ao trabalhar com objetos Range (faixa), tenha em mente os seguintes pontos:

5 O seu VBA não precisa *selecionar* uma faixa para trabalhar com ela.

5 Se o seu código selecionar uma faixa, a planilha dele deve estar ativa.

5 O gravador de macro nem sempre gera o código mais eficiente.

Frequentemente, você pode criar a sua macro usando o gravador e depois, editar o código, para torná-lo mais eficaz.

ExcelVBABokks.indb 211

15/02/2013 13:59:53

212 Parte III: Conceitos de Programação 5 É uma boa ideia usar faixas nomeadas em seu código VBA. Por exemplo, usar Range(“Total”) é melhor que usar Range(“D45”).

Nesse caso, se você acrescentar uma linha acima da linha 45,

precisará modificar a macro para que ela use o endereço de

faixa correto (D46). Veja que você nomeia uma faixa de células, escolhendo Fórmulas↔Nomes Definidos↔Definir Nome.

5 Ao rodar a sua macro que trabalha na seleção atual de faixas, o usuário poderia selecionar colunas ou linhas inteiras. Na

maioria dos casos, você não quer fazer loop através de cada célula na seleção (isso poderia demorar muito). A sua macro deveria criar um subconjunto da seleção, consistindo apenas de células que não estão em branco.

5 O Excel permite múltiplas seleções. Por exemplo, você pode

selecionar uma faixa, pressionar Ctrl e selecionar outra faixa com o seu mouse. O seu código pode ser testado nesses casos,

de modo que você tome as ações apropriadas.

Os exemplos nesta seção, que estão disponíveis no site deste livro, demonstram estes pontos.

Se você prefere digitar esses exemplos por si próprio, pressione Alt + F11

BRE-SE

para ativar o VBE. Depois, insira um módulo VBA e digite o código.

LEM

Assegure-se de que a pasta de trabalho esteja configurada adequadamente. Se o exemplo usar duas planilhas chamadas Sheet1 e Sheet2, assegure-se de que a pasta de trabalho tenha as planilhas com esses nomes.

Copiando uma faixa

Uma das atividades em Excel considerada a mais preferida de todos os tempos é copiar uma faixa. Quando você liga o gravador de macro e copia uma faixa a partir de A1:A5 para B1:B5, obtém esta macro VBA: Sub CopyRange() Range("A1:A5").Select

```
Selection.Copy
```

```
Range("B1").Select
```

```
ActiveSheet.Paste
```

```
Application.CutCopyMode = False
```

```
End Sub
```

Observe a última declaração. Ela foi gerada pressionando Esc, o que cancela a marcha das formigas que aparece na planilha quando você copia uma faixa.

Esta macro funciona bem, mas você pode copiar uma faixa com mais eficiência. É possível produzir o mesmo resultado com a seguinte macro de uma linha, a qual não seleciona quaisquer células: Sub CopyRange2()

```
Range("A1:A5").CopyRange("B1")
```

```
End Sub
```

ExcelVBABokks.indb 212

15/02/2013 13:59:53

	A	B	C	D
1	Data	Unidades	Valor	
2	15/nov	132	R\$ 2.727	
3	16/nov	143	R\$ 154	
4	17/nov	133	R\$ 109	
5	18/nov	169	R\$ 614	
6	19/nov	102	R\$ 2.744	
7	20/nov	143	R\$ 5.164	
8	21/nov	109	R\$ 4.314	
9	22/nov	122	R\$ 4.448	
10	23/nov	156	R\$ 4.657	
11	24/nov	187	R\$ 6.989	
12	25/nov	140	R\$ 2.014	
13	26/nov	132	R\$ 1.070	
14				

Capítulo 14: Exemplos de Programação em VBA 213

Este procedimento tem a vantagem de que o método Copy pode usar um argumento que especifica o destino. Eu descobri isso consultando o sistema de ajuda do VBA. Este exemplo também demonstra que o gravador de macro nem sempre gera o código mais eficiente.

Copiando uma faixa de tamanho variável

Em muitos casos, você precisa copiar uma faixa de células, mas não conhece as dimensões exatas de linha e coluna. Por exemplo, você poderia ter uma pasta de trabalho que controla as vendas semanais. A quantidade de linhas muda quando você acrescenta novos dados.

A Figura 14-1 mostra uma faixa em uma planilha. Essa faixa consiste de várias linhas e uma série de linhas pode mudar diariamente. Pelo fato de você não saber o endereço exato da faixa em determinada ocasião, escrever uma macro para copiá-la pode ser desafiador. Está preparado para o desafio?

Figura 14-1:

Essa faixa

pode consistir de

qualquer

quantidade

de linhas.

A seguinte macro demonstra como copiar essa faixa de Sheet1 para Sheet2 (começando na célula A1). Ela usa a propriedade CurrentRegion, a qual retorna um objeto Ranger que corresponde ao bloco de células em torno de uma célula em especial. Nesse caso, tal célula é A1.

```
Sub CopyCurrentRegion()
```

```
Range("A1").CurrentRegion.Copy
```

```
Sheets("Sheet2").Select
```

```
Range("A1").Select
```

```
ActiveSheet.Paste
```

```
Sheets("Sheet1").Select
```

```
Application.CutCopyMode = False
```

```
End Sub
```

ExcelVBABokks.indb 213

15/02/2013 13:59:54

214 Parte III: Conceitos de Programação Usar a propriedade

CurrentRegion é equivalente a escolher Página

Inicial⇨Edição⇨Localizar e Selecionar⇨Ir para Especial (que exibe a caixa de diálogo Ir para Especial) e escolher a opção Região Atual.

Para ver como isso funciona, grave as suas ações enquanto executa esse comando. Normalmente, Região Atual consiste de um bloco retangular de células rodeado por uma ou mais linhas ou colunas em branco.

É possível tornar essa macro ainda mais eficiente, não selecionando o destino. A macro a seguir tem a vantagem de que o método Copy pode usar um argumento para a faixa de destino: Sub

```
CopyCurrentRegion2()
```

```
Range("A1").CurrentRegion.Copy _
```

```
Sheets("Sheet2").Range("A1")
```

```
Application.CutCopyMode = False
```

```
End Sub
```

Selecionando ao final de uma linha ou coluna

Provavelmente, você tem o costume de usar combinações de teclas, como Ctrl + Shift + Seta para a direita e Ctrl + Shift + Seta para baixo para selecionar uma faixa que consiste de tudo, a partir da célula ativa ao final de uma linha ou uma coluna. Sem surpresas, você pode escrever macros que executam esses tipos de seleção.

Você pode usar a propriedade CurrentRegion para selecionar um bloco inteiro de células. Mas, e se você quiser selecionar, digamos, uma coluna de um bloco de células? Felizmente, o VBA pode acomodar esse tipo de ação. O seguinte procedimento VBA seleciona a faixa iniciando na célula ativa e se estendendo para baixo. Depois de selecionar a faixa, você pode fazer o que quiser com ela — copiá-la, movê-la, formatá-la e assim por diante.

```
Sub SelectDown()
```

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

```
End Sub
```

É possível fazer manualmente esse tipo de seleção: Selecione a primeira célula, mantenha pressionada a tecla Shift, pressione End e depois, pressione Seta para baixo.

Este exemplo usa o método End do objeto ActiveCell, o qual retorna um objeto Range. O método End toma um argumento que pode ser qualquer das seguintes constantes:

5 x1UP

5 x1Down

5 xlToLeft

5 xlToRight

ExcelVBABokks.indb 214

15/02/2013 13:59:54

Capítulo 14: Exemplos de Programação em VBA 215

Tenha em mente que não é necessário selecionar uma faixa antes de fazer alguma coisa com ela. A seguinte macro aplica a formatação em negrito a uma faixa de tamanho variável (coluna única) sem selecionar a faixa: Sub MakeBold()

```
Range(ActiveCell, ActiveCell.End(xlDown)) _
```

```
.Font.Bold = True
```

```
End Sub
```

Selecionando uma linha ou coluna

O seguinte procedimento demonstra como selecionar a coluna que contém a célula ativa. Ele usa a propriedade `EntireColumn`, a qual retorna um objeto `Range` que consiste de uma coluna inteira: Sub `SelectColumn()`

```
ActiveCell.EntireColumn.Select
```

```
End Sub
```

Como você poderia esperar, o VBA também oferece uma propriedade `EntireRow`, a qual retorna um objeto `Range` que consiste de uma linha inteira.

Movendo uma faixa

Você move uma faixa recortando-a para a Área de Transferência e depois colando-a em outra área. Se gravar as suas ações enquanto

executa uma operação de mover, o gravador de macro gera um código como o seguinte: Sub MoveRange()

```
Range("A1:C6").Select
```

```
Selection.Cut
```

```
Range("A10").Select
```

```
ActiveSheet.Paste
```

```
End Sub
```

Como com a cópia do exemplo, anteriormente neste capítulo, essa não é a maneira mais eficiente de mover uma faixa de células. Na verdade, você pode mover uma faixa com uma única declaração VBA, assim: Sub MoveRange2()

```
Range("A1:C6").Cut Range("A10")
```

```
End Sub
```

ExcelVBABokks.indb 215

15/02/2013 13:59:54

216 Parte III: Conceitos de Programação Esta macro tem a vantagem de que o método Cut pode usar um argumento que especifica o destino. Observe ainda que a faixa não foi selecionada. O indicador de célula permanece em sua posição original.

Como fazer loop eficientemente

através de uma faixa

Muitas macros executam uma operação em cada célula de uma faixa, ou poderiam executar ações selecionadas, com base no conteúdo de cada célula. Geralmente, essas células incluem um loop For-Next que processa cada célula na faixa.

O seguinte exemplo demonstra como fazer loop através de uma faixa de células. Neste caso, a faixa é a seleção atual. Um objeto variável, chamado Cell, refere-se à célula sendo processada. Dentro da loop For-Next, a única declaração avalia a célula e aplica formatação em negrito se a célula contiver um valor positivo.

```
Sub ProcessCells()
```

Dim Cell As Range

For Each Cell In Selection

If Cell.Value > 0 Then Cell.Font.Bold = True

Next Cell

End Sub

Este exemplo funciona, mas e se a seleção consistir de uma coluna ou linha inteira? Isso não é incomum, pois o Excel permite que você realize operações em colunas ou linhas inteiras. Em tal caso, a macro parece demorar para sempre, pois ela faz loops através de cada célula (todas as 1.048.576) na coluna – mesmo as células em branco. Para tornar a macro mais eficiente, você precisa de uma maneira para processar apenas as células que não estão em branco.

A seguinte rotina faz exatamente isso, usando o método SpecialCells (para detalhes específicos sobre os argumentos dele, consulte o sistema de Ajuda VBA). Esta rotina usa a palavra chave Set para criar dois novos objetos Range: o subconjunto da seleção, que consiste de células com constantes e o subconjunto da seleção que consiste de células com fórmulas. A rotina processa cada um desses subconjuntos, com o efeito líquido de pular todas as células em branco. Bem esperto, não é?

Sub SkipBlanks()

Dim ConstantCells As Range

Dim FormulaCells As Range

Dim cell As Range

‘ Ignore erros

On Error Resume Next

ExcelVBABokks.indb 216

15/02/2013 13:59:55

Capítulo 14: Exemplos de Programação em VBA 217

‘ Processe as constantes

Set ConstantCells = Selection _
.SpecialCells(xlConstants)

For Each cell In ConstantCells

If cell.Value > 0 Then

cell.Font.Bold = True

End If

Next cell

‘ Processe as fórmulas

```
Set FormulaCells = Selection
```

```
.SpecialCells(xlFormulas)
```

```
For Each cell In FormulaCells
```

```
If cell.Value > 0 Then
```

```
cell.Font.Bold = True
```

```
End If
```

```
Next Cell
```

```
End Sub
```

O procedimento SkipBlanks funciona na mesma velocidade, independente do que você seleciona. Por exemplo, você pode selecionar a faixa, todas as colunas na faixa, todas as linhas na faixa ou mesmo toda a planilha. É um grande aperfeiçoamento sobre o procedimento ProcessCells, apresentado anteriormente nesta seção.

Veja que eu uso a seguinte declaração neste código:

```
On Error Resume Next
```

Esta declaração diz ao Excel para ignorar quaisquer erros que ocorram e simplesmente processar a seguinte declaração (veja no Capítulo 12 uma discussão sobre como lidar com erros). Esta declaração é necessária, porque o método SpecialCells produz um erro se nenhuma das células se qualificar.

Usar o método SpecialCells é equivalente a escolher o comando Página inicial⇒Edição⇒Localizar e Seleccionar⇒Ir para Especial e seleccionar a opção Constantes ou a opção Fórmulas. Para ter uma noção de como isso funciona, grave as suas ações enquanto executa tal comando e selecciona as várias opções.

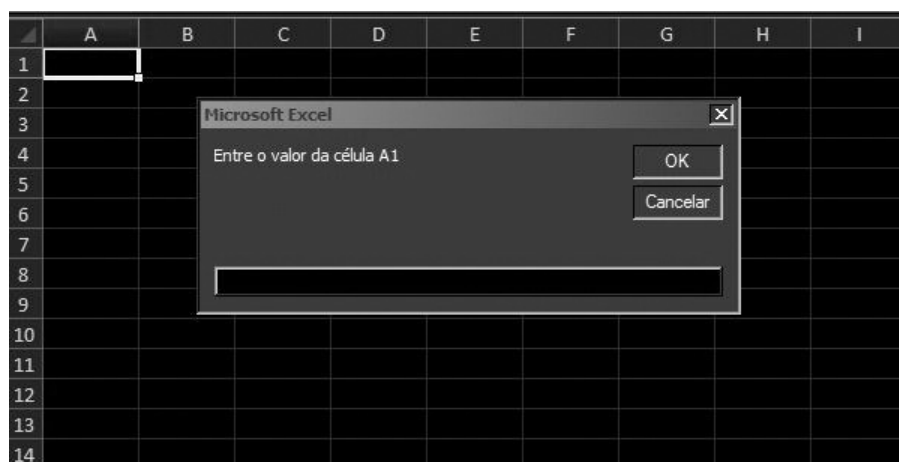
Como fazer loop eficientemente

através de uma faixa (Parte II)

E agora, a continuação. Esta seção demonstra uma outra forma de processar células de uma maneira eficiente. Ela tem a vantagem da propriedade UsedRange (Faixa Usada) — a qual retorna um objeto Range que consiste apenas da área usada da planilha. Ela também usa o método Intersect, que retorna um objeto Range que consiste de células que duas faixas têm em comum.

ExcelVBABokks.indb 217

15/02/2013 13:59:55



218 Parte III: Conceitos de Programação

Eis uma variação do procedimento SkipBlanks da seção anterior: Sub SkipBlanks2()

Dim WorkRange As Range

Dim cell As Range

Set WorkRange = Intersect _

(Selection, ActiveSheet.UsedRange)

For Each cell In WorkRange

If cell.Value > 0 Then

cell.Font.Bold = True

End If

Next cell

End Sub

O objeto variável WorkRange consiste de células que são comuns à seleção do usuário, e a faixa usada da planilha. Portanto, se toda uma coluna estiver selecionada, WorkRange contém apenas as células que estão dentro da área usada da planilha. Rápido e eficiente, sem ciclos de CPU desperdiçados no processamento de células que estão fora da área usada na planilha.

Solicitando o valor de uma célula

Conforme mostrado na Figura 14-2, você pode usar a função InputBox do VBA para obter um valor do usuário. Depois, você pode inserir aquele valor em uma célula. O seguinte procedimento demonstra

como pedir um valor ao usuário e colocar o valor na célula A1 da planilha ativa, usando apenas uma declaração: Sub GetValue()

```
Range("A1").Value = InputBox _
```

```
("Entre o valor da célula A1")
```

```
End Sub
```

Figura 14-2:

Use a

função

InputBox do

VBA para

obter um

valor do

usuário.

ExcelVBABokks.indb 218

15/02/2013 13:59:55

Capítulo 14: Exemplos de Programação em VBA 219

Se experimentar este exemplo, você descobrirá que clicar o botão Cancelar na InputBox apaga o valor atual na célula A1. Apagar os dados do usuário não é uma prática muito boa de programação. A seguinte macro demonstra uma abordagem melhor: usar uma variável (x) para armazenar o valor fornecido pelo usuário. Se o valor não estiver vazio (isto é, se o usuário não cancelou), o valor de x é colocado na célula A1.

Caso contrário, nada acontece.

```
Sub GetValue2()
```

Dim x as Variant

x = InputBox(“Entre o valor da célula A1”)

If x < > “ “ Then Range(“A1”).Value = x

End Sub

A variável x é definida como um tipo de dados Variant (variante), pois ela poderia ser um número ou uma string vazia (se o usuário cancelar).

Determinando o tipo de seleção

Se você definir a sua macro com uma seleção de faixa, a macro deve ser capaz de determinar se uma faixa está, de fato, selecionada. Se alguma outra coisa que não uma faixa estiver selecionada (tal como um gráfico ou uma figura), provavelmente a macro explodirá. O seguinte procedimento usa a função TypeName para identificar o tipo de objeto que está selecionado no momento: Sub SelectionType()

MsgBox TypeName(Selection)

End Sub

Se um objeto Range estiver selecionado, a MsgBox exibe Range. Se a sua macro só funcionar com faixas, você pode usar uma declaração If para garantir que uma faixa seja selecionada. Este exemplo exibe uma mensagem e promove a saída do procedimento se a seleção atual não for um objeto Range: Sub CheckSelection()

If TypeName(Selection) < > “Range” Then

MsgBox “Selecione uma faixa.”

Exit Sub

End If

‘ ... [Other statements go here]

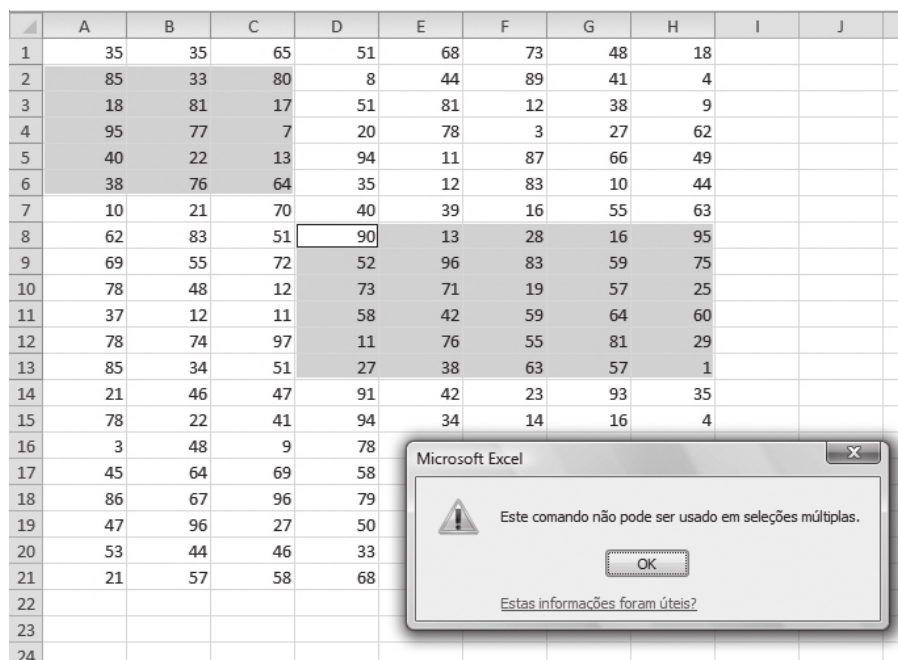
End Sub

Identificando uma seleção múltipla

Como você sabe, o Excel permite múltiplas seleções, pressionando Ctrl enquanto você escolhe objetos ou faixas. Isso pode causar problemas com algumas macros. Por exemplo, você não pode copiar uma seleção múltipla que consiste de células não adjacentes. Se tentar fazê-lo, o Excel o repreende com a mensagem mostrada na Figura 14-3.

ExcelVBABokks.indb 219

15/02/2013 13:59:55



O Excel não

gosta se

você tentar

copiar uma

seleção

múltipla.

A seguinte macro demonstra como determinar se o usuário fez uma seleção múltipla, assim a sua macro pode tomar a atitude apropriada:

Sub MultipleSelection()

If Selection.Areas.Count > 1 Then

MsgBox “Seleções múltiplas não permitidas.”

Exit Sub

End If

‘ ... [Outras declarações entram aqui

End Sub

Este exemplo usa o método Areas, que retorna uma coleção de todos os objetos na seleção. A propriedade Count retorna o número de objetos na coleção.

Mudando as Configurações do Excel

Algumas das macros mais úteis são simples procedimentos que mudam uma ou mais configurações do Excel. Por exemplo, se você acessa com frequência a caixa de diálogo Opções do Excel para alterar uma configuração, trata-se de uma boa oportunidade para poupar tempo

com uma simples macro.

Esta seção apresenta dois exemplos que mostram como mudar configurações no Excel. Você pode aplicar os princípios gerais demonstrados por estes exemplos em outras operações que alteram configurações.

ExcelVBABokks.indb 220

15/02/2013 13:59:56

Capítulo 14: Exemplos de Programação em VBA 221

Mudando configurações Booleanas

Como um interruptor de luz, uma configuração *Booleana* está ligada ou desligada (ativada ou desativada). Por exemplo, você poderia querer criar uma macro que ativasse e desativasse a exibição de quebra de página de planilha. Depois que você imprime ou visualiza uma planilha, o Excel exibe linhas pontilhadas para indicar as quebras de página.

Algumas pessoas (incluindo o autor) acha essas linhas pontilhadas muito desagradáveis. Infelizmente, a única maneira de se livrar da exibição de quebra de página é abrir a caixa de diálogo Opções do Excel, clicar na guia Avançado e rolar para baixo, até encontrar a caixa de verificação Mostrar Quebras de Página. Se você ativar o gravador de macro ao mudar essa opção, o Excel gera o seguinte código: `ActiveSheet.DisplayPageBreaks = False`

Por outro lado, se as quebras de página não estiverem visíveis quando você gravar a macro, o Excel gera o seguinte código:

```
ActiveSheet.DisplayPageBreaks = True
```

Isso pode levá-lo a suspeitar que precisa de duas macros: uma para ativar a exibição de quebra de página e uma para desativar. Não é verdade. O seguinte procedimento usa o operador Not (não), o qual troca True para False e False para True. Executar o procedimento `TogglePageBreaks` é uma maneira simples de ativar a exibição de quebra de página de True para False e de False para True: `Sub TogglePageBreaks()`

On Error Resume Next

```
ActiveSheet.DisplayPageBreaks = Not _
```

```
ActiveSheet.DisplayPageBreaks
```

```
End Sub
```

A primeira declaração ignora um erro que acontece se a planilha ativa for uma planilha de gráfico (planilhas de gráfico não exibem quebras de página).

Você pode usar esta técnica para alternar quaisquer configurações que tenham valores Booleanos (Verdadeiro ou Falso).

Mudando configurações não Booleanas

Use uma estrutura Select Case em configurações não Booleanas. Este exemplo alterna o modo de cálculo entre manual e automático e exibe uma mensagem indicando o modo atual: ExcelVBABokks.indb 221

15/02/2013 13:59:56

222 Parte III: Conceitos de Programação Sub ToggleCalcMode()

```
Select Case Application.Calculation
```

```
Case xlManual
```

```
Application.Calculation = xlCalculationAutomatic
```

```
MsgBox “Modo de cálculo Automatico”
```

```
Case xlAutomatic
```

```
Application.Calculation = xlCalculationManual
```

```
MsgBox “Modo de cálculo Manual”
```

```
End Select
```

```
End Sub
```

É possível adaptar esta técnica para alterar outras configurações não Booleanas.

Trabalhando com Gráficos

Os gráficos são repletos de objetos diferentes, portanto, manipular gráficos com VBA pode ser um pouco desafiador. O desafio aumenta com o Excel 2007, porque a Microsoft resolveu omitir a gravação de macros em todas as coisas novas e extravagantes de formatação de gráfico. Felizmente, esse problema sério foi corrigido no Excel 2010.

Eu disparei o Excel 2010, entrei com alguns números em A1:A3 e selecionei aquela faixa. Depois, liguei o gravador de macro e criei um gráfico básico de coluna com três pontos de dados. Apaguei a legenda do gráfico e acrescentei um efeito sombreado às colunas. Eis a macro:

```
Sub Macro1() ActiveSheet.Shapes.AddChart.Select
```

```
ActiveChart.ChartType = xlColumnClustered
```

```
ActiveChart.SetSourceData Source:=Range _
```

```
("Sheet1' !$A$1:$A$3")
```

```
ActiveChart.Legend.Select
```

```
Selection.Delete
```

```
ActiveSheet.ChartObjects("Chart 1").Activate
```

```
ActiveChart.SeriesCollection(1).Select
```

```
Selection.Format.Shadow.Type = msoShadow21
```

```
End Sub
```

Se você gravar essa macro em Excel 2007, a última declaração (a qual

aplica a sombra) nem ao menos é gerada. Este é apenas um exemplo de como o Excel 2007 ignora os comandos de formatação de gráfico ao gravar uma macro.

A propósito, esta macro provavelmente gerará um erro, pois ela faz código sólido com o nome do gráfico na macro. Quando você roda essa macro, o gráfico criado não é necessariamente nomeado como Chart 1. Se por acaso você tiver um gráfico chamado Chart 1, a formatação de sombreado será aplicada a ele — não ao que a macro criou. Da mesma forma, a faixa de ExcelVBABokks.indb 222

15/02/2013 13:59:56

Capítulo 14: Exemplos de Programação em VBA

223

dados do gráfico é de código sólido, portanto, você não usaria isto em uma macro de criação de gráfico de objetivos gerais.

Entretanto, examinar o código gravado revela algumas coisas que podem ser úteis ao escrever as suas próprias macros relacionadas com gráfico. Se estiver curioso, eis uma versão feita à mão daquela macro que cria um gráfico a partir de uma faixa selecionada: Sub CreateAChart()

Dim ChartData As Tange

Dim ChartShape As Shape

Dim NewChart As Chart

‘ Create object variables

Set ChartData = ActiveWindow.RangeSelection

```
Set ChartShape = ActiveSheet.Shapes.AddChart
```

```
Set NewChart = ChartShape.Chart
```

```
‘ Adjust the chart
```

```
With NewChart
```

```
.ChartType = xlColumnClustered
```

```
.SetSourceData Source:=Range(ChartData.Address)
```

```
.Legend.Delete
```

```
.SeriesCollection(1).Format.Shadow.Type = _
```

```
msoShadow21
```

```
End With
```

```
End Sub
```

Se precisar escrever macros VBA que manipulem gráficos, você precisa entender alguma terminologia. Um *gráfico embutido* em uma planilha é um objeto `ChartObject`. Você pode ativar um `ChartObject` exatamente como ativa uma planilha. A seguinte declaração ativa o `ChartObject` chamado Chart 1: `ActiveSheet.ChartObjects(“Chart 1”).Activate`

Depois de ativar o gráfico, você pode referenciá-lo em seu código VBA como o `ActiveChart`. Se o gráfico estiver em uma planilha de gráfico separada, ele se torna o gráfico ativo assim que você ativar a sua respectiva planilha.

Um `ChartObject` também é um `Shape`, que pode ser um pouco confuso.

Na verdade, quando o seu código VBA criar um gráfico, ele começa

O

acrescentando um novo Shape. Você também pode ativar um gráfico,
P A P

selecionando o objeto Shape que contém o gráfico:

```
ActiveSheet.Shapes("Chart 1").Select
```

Eu prefiro usar o objeto ChartObject em meu código, apenas para deixar perfeitamente claro que estou trabalhando com um gráfico.

ExcelVBABokks.indb 223

15/02/2013 13:59:57

224 Parte III: Conceitos de Programação Quando você clicar um gráfico embutido, na verdade o Excel seleciona um objeto *dentro* do objeto ChartObject. Você pode selecionar o próprio **CUIDADO!**

ChartObject, pressionando Ctrl enquanto clica o gráfico embutido.

Modificando o tipo de gráfico

Eis uma declaração confusa para você: Um ChartObject age como um contêiner para um objeto Chart. Leia isso algumas vezes e poderá, de fato, fazer sentido.

Para modificar um gráfico com VBA, não é preciso ativar o gráfico.

Ao invés disso, o método Chart pode retornar o gráfico contido no ChartObject. Você já está bem confuso? Os dois procedimentos seguintes têm o mesmo efeito — eles mudam o gráfico chamado Chart 1 para uma área de gráfico. O primeiro procedimento ativa o primeiro gráfico e depois, trabalha com o gráfico ativo. O segundo procedimento não ativa o gráfico. Ao invés, ele usa a propriedade Chart para retornar o objeto Chart contido no objeto ChartObject.

```
Sub ModifyChart1()
```

```
ActiveSheet.ChartObjects("Chart 1").Activate
```



```
ActiveChart.Type = xlArea
```

```
End sub
```

```
Sub ModifyChart2()
```

```
ActiveSheet.ChartObjects("Chart 1") _
```

```
.Chart.Type = xlArea
```

```
End Sub
```

Fazendo Looping através da

coleção ChartObjects

Este exemplo muda o tipo de gráfico de cada gráfico embutido na planilha ativa. O procedimento usa um loop For-Next para circular através de cada objeto na coleção ChartObjects, acessar o objeto Chart em cada um e alterar a sua propriedade Type.

```
Sub ChartType()
```

```
Dim cht As ChartObject
```

```
For Each cht In ActiveSheet.ChartObjects
```

```
cht.Chart.Type = xlArea
```

```
Next cht
```

```
End Sub
```

Capítulo 14: Exemplos de Programação em VBA

225

A seguinte macro executa a mesma função, mas funciona em todas as planilhas de gráfico na pasta de trabalho ativa:

```
Sub ChartType2()
```

```
Dim cht As ChartType2
```

```
For Each cht In ActiveWorkbook.Charts
```

```
cht.Type = xlArea
```

```
Next cht
```

```
End Sub
```

Modificando propriedades Chart

O seguinte exemplo muda a fonte da legenda em todos os gráficos na planilha ativa. Ele usa um loop For-Next para processar todos os objetos ChartObject:

```
Sub LegendMod()
```

```
Dim cht As ChartObject
```

```
For Each cht In ActiveSheet.ChartObjects
```

With cht.Chart.Legend.Font

.Name = “Calibri”

.FontStyle = “Bold”

.Size = 12

End With

Next cht

End Sub

Observe que o objeto Font está contido no objeto Legend, o qual está contido no objeto Chart, que está contido na coleção ChartObjects.

Agora você entende porque é chamada de *hierarquia de objeto*?

Aplicando formatação de gráfico

Este exemplo aplica vários tipos diferentes de formatação ao gráfico ativo.

Eu criei esta macro gravando minhas ações enquanto formatava um gráfico. Depois, limpei o código gravado, removendo as linhas irrelevantes.

Sub ChartMods()

ActiveChart.Type = xlArea

ActiveChart.ChartArea.Font.Name = “Calibri”

ActiveChart.ChartArea.Font.FontStyle = “Regular”

ActiveChart.ChartArea.Font.Size = 9

ActiveChart.PlotArea.Interior.ColorIndex = xlNone

ActiveChart.Axes(xlValue).TickLabels.Font.Bold = True

ActiveChart.Axes(xlCategory).TickLabels.Font.Bold = _
True

ActiveChart.Legend.Position = xlBottom

End Sub

ExcelVBABokks.indb 225

15/02/2013 13:59:57

226 Parte III: Conceitos de Programação Você deve ativar um gráfico antes de executar a macro ChartMods.

Ative um gráfico embutido clicando-o. Para ativar um gráfico em uma planilha de gráfico, ative a planilha de gráfico.

Para garantir que um gráfico seja selecionado, é possível acrescentar uma declaração para determinar se um gráfico está ativo. Eis a macro modificada, que exibe uma mensagem (e termina) se um gráfico não estiver ativado: Sub ChartMods2()

If ActiveChart Is Nothing Then

MsgBox “Activate a chart.”

Exit Sub

End If

ActiveChart.Type = xlArea

ActiveChart.ChartArea.Font.Name = “Calibri”

ActiveChart.ChartArea.Font.FontStyle = “Regular”

ActiveChart.ChartArea.Font.Size = 9

ActiveChart.PlotArea.Interior.ColorIndex = xlNone

ActiveChart.Axes(xlValue).TickLabels.Font.Bold = True

```
ActiveChart.Axes(xlCategory).TickLabels.Font.Bold = _
```

```
True
```

```
ActiveChart.Legend.Position = xlBottom
```

```
End Sub
```

Eis uma outra versão que usa a construção With-End With para poupar alguma digitação e tornar o código um pouco mais claro. Novamente, estou me adiantando a mim mesmo. Folheie algumas páginas para ler sobre a estrutura de With End-With.

```
Sub ChartMods3()
```

```
If ActiveChart Is Nothing Then
```

```
MsgBox “Activate a chart.”
```

```
Exit Sub
```

```
End If
```

```
With ActiveChart
```

```
.Type = xlArea
```

```
.ChartArea.Font.Name = “Calibri”
```

```
.ChartArea.Font.FontStyle = “Regular”
```

```
.ChartArea.Font.Size = 9
```

```
.PlotArea.Interior.ColorIndex = xlNone
```

```
.Axes(xlValue).TickLabels.Font.Bold = True
```

```
.Axes(xlCategory).TickLabels.Font.Bold = True
```

```
.Legend.Position = xlBottom
```

```
End With
```

```
End Sub
```

Quando se trata de usar VBA para trabalhar com gráficos, esta curta seção mal arranhou a superfície. Claro que há muito mais sobre isso, mas pelo menos esta introdução básica irá encaminhá-lo na direção certa.

ExcelVBABokks.indb 226

15/02/2013 13:59:57

Capítulo 14: Exemplos de Programação em VBA

227

Dicas de Velocidade do VBA

VBA é rápido, mas nem sempre rápido o suficiente (os programas de computador nunca são rápidos o suficiente). Esta seção apresenta alguns exemplos de programação que podem ser usados para aumentar a velocidade de suas macros.

Desativando a atualização de tela

Ao executar uma macro, você pode sentar e observar na tela toda a ação que acontece na macro. Ainda que fazer isso possa ser instrutivo, depois de conseguir fazer a macro funcionar adequadamente, geralmente é aborrecido e pode desacelerar consideravelmente o fluxo de sua macro. Felizmente, você pode desativar a atualização de tela que normalmente acontece quando você executa uma macro. Para desativar a atualização de tela, use a seguinte declaração:

```
Application.ScreenUpdating = False
```

Se você quiser que o usuário veja o que está acontecendo em qualquer momento durante a macro, use a seguinte declaração para retornar a ativação de tela:

```
Application.ScreenUpdating = True
```

Para demonstrar a diferença em velocidade, execute esta simples macro, a qual preenche uma faixa com números:

```
Sub FillRange()
```

```
Dim r As Long, c As Long
```

```
Dim Number As Long
```

```
Number = 0
```

```
For r = 1 To 50
```

```
For c = 1 To 50
```


Number = Number + 1

Cells(r, c).Select

Cells(r, c).Value = Number

Next c

Next r

End Sub

Você vê cada célula sendo selecionada e o valor sendo inserido. Agora, insira a seguinte instrução no início do procedimento e execute novamente.

Application.ScreenUpdating = False

ExcelVBABokks.indb 227

15/02/2013 13:59:58

228 Parte III: Conceitos de Programação A faixa é preenchida muito mais depressa e você não vê o resultado final até que a macro tenha acabado de rodar.

Ao depurar código, às vezes a execução do programa termina em algum **DICA**

lugar no meio, sem ter retornado a atualização de Tela (é, isso acontece comigo também). Às vezes, isso leva a janela de aplicativo do Excel ficar completamente sem resposta. A forma de sair dessa posição congelada é simples: volte para o VBE e execute a seguinte declaração na janela Verificação imediata:

`Application.ScreenUpdating = True`

Desativando o cálculo automático

Se você tiver uma planilha com muitas fórmulas complexas, pode achar que é possível agilizar consideravelmente as coisas, configurando o modo de cálculo para manual enquanto a sua macro está executando.

Quando a macro terminar, configure o cálculo de volta para automático.

A seguinte declaração configura o modo de cálculo do Excel para manual: `Application.Calculation = xlCalculationManual`

Execute a seguinte declaração para configurar o modo de cálculo para automático:

`Application.Calculation = xlCalculationAutomatic`

Se o seu código usar células com resultados de fórmulas, lembre-se de **CUIDADO!**

que desativar o cálculo significa que as células não serão recalculadas, a menos que você diga, explicitamente, ao Excel para fazê-lo!

Eliminando aquelas inoportunas

mensagens de alerta

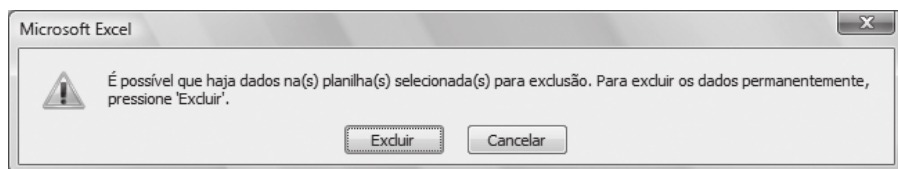
Como você sabe, uma macro pode executar automaticamente uma série de ações. Em muitos casos, você pode iniciar uma macro e depois, circular pela sala de correspondência enquanto o Excel faz isso.

Entretanto, algumas operações do Excel exibem mensagens que

requerem uma resposta humana. Por exemplo, se a sua macro apagar uma planilha que não está vazia, o seu código dá uma parada, enquanto o Excel espera pela sua resposta à mensagem mostrada na Figura 14-4.

Esses tipos de mensagem significam que você não pode deixar o Excel sozinho enquanto ele executa a sua macro — a menos que você conheça o truque secreto.

15/02/2013 13:59:58



Capítulo 14: Exemplos de Programação em VBA 229

Figura 14-4:

Você pode
instruir o
Excel a não
exibir esses
tipos de
alertas
enquanto
executa
uma macro.

O truque secreto: para evitar essas mensagens de alerta, insira a declaração a seguir em sua macro.

```
Application.DisplayAlerts = False
```

O Excel executa a operação padrão para esses tipos de mensagens. No caso de apagar uma planilha, a operação padrão é Excluir (que é exatamente o que você quer que aconteça). Se não tiver certeza de qual é a operação padrão, faça um teste e veja o que acontece.

Quando o procedimento terminar, automaticamente o Excel reconfigura a propriedade `DisplayAlerts` para `True` (a sua posição normal). Se você precisar retornar a ativação do alerta antes do procedimento terminar, use esta declaração: `Application.DisplayAlerts`

= True

Simplificando referências de objeto

Como provavelmente você já sabe, referências a objetos podem se tornar muito longas. Por exemplo, uma referência totalmente qualificada a um objeto Range pode parecer como isto:

```
Workbooks("MyBook.xlsx").Worksheets("Sheet1"). _
```

```
.Range("InterestRate")
```

Se a sua macro usa essa faixa com frequência, você pode querer criar um objeto variável, usando o comando Set. Por exemplo, a seguinte declara-

ção designa esse objeto Range a um objeto variável chamado Rate: Set
Rate = Workbooks("MyBook.xlsx") _

```
.Worksheets("Sheet1").Range("InterestRate")
```

ExcelVBABokks.indb 229

15/02/2013 13:59:58

230 Parte III: Conceitos de Programação Depois de definir este objeto variável, você pode usar a variável Rate ao invés da referência longa. Por exemplo, é possível mudar o valor da célula chamada InterestRate: Rate.Value = .085

Isto é muito mais fácil de digitar (e entender) que a seguinte declaração: Workbooks("MyBook.xlsx").Worksheets("Sheet1"). _

```
Range("InterestRate") = .085
```

Além de simplificar a sua codificação, usar objetos variáveis também agiliza consideravelmente as suas macros. Depois de criar objetos variáveis, tenho visto algumas macros rodar duas vezes mais depressa que antes.

Declarando tipos de variáveis

Normalmente, você não precisa se preocupar com o tipo de dados que designa a uma variável. O Excel lida com todos os detalhes por trás das cenas para você. Por exemplo, se tiver uma variável chamada MyVar, você pode designar um número de qualquer tipo àquela

variável. Pode até designar uma string de texto a ela, mais adiante no procedimento.

Mas, se você quiser que os seus procedimentos rodem o mais rápido
BRE-SE

possível (e evitem alguns problemas potencialmente desagradáveis),
LEM

diga ao Excel qual tipo de dados serão designados a cada uma de suas variáveis. Isso é conhecido como *declarar* um tipo da variável (para detalhes completos, consulte o Capítulo 7). Acostume-se a declarar todas as variáveis que você usa.

Em geral, você deveria usar o tipo de dados que requer o menor número de bytes que, no entanto, lidam com todos os dados designados a ele. Quando o VBA trabalha com dados, a velocidade de execução depende do número de bytes que o VBA tem à sua disposição. Em outras palavras, quanto menos bytes os dados usam, mais depressa o VBA pode acessar e manipular os dados.

Se você usa um objeto variável (conforme descrito na seção anterior), pode declarar a variável como um tipo de objeto especial. Eis um exemplo: Dim Rate as Range

```
Set Rate = Workbooks("MyBook.xlsx") _  
.Worksheets("Sheet1").Range("InterestRate")
```

ExcelVBABokks.indb 230

15/02/2013 13:59:59

Capítulo 14: Exemplos de Programação em VBA 231

Como usar a estrutura With-End With

Você precisa configurar um número de propriedades a um objeto? O

DICA

seu código roda mais depressa se você usar a estrutura With-End With.

Um benefício adicional é que o seu código pode ficar mais fácil de ler.

O seguinte código não usa With-End With:

```
Selection.HorizontalAlignment = xlCenter
```

```
Selection.VerticalAlignment = xlCenter
```

```
Selection.WrapText = True
```

```
Selection.Orientation = 0
```

```
Selection.ShrinkToFit = False
```

```
Selection.MergeCells = False
```

Eis o mesmo código, reescrito para usar With-End With:

```
With Selection
```

```
.HorizontalAlignment = xlCenter
```

```
.VerticalAlignment = xlCenter
```

```
.WrapText = True
```

```
.Orientation = 0
```

```
.ShrinkToFit = False
```

```
.MergeCells = False
```

```
End With
```

Se esta estrutura parece familiar, provavelmente é porque o gravador de macro usa With-End With sempre que pode. E eu apresentei um outro exemplo anteriormente neste capítulo.

15/02/2013 13:59:59

232 **Parte III: Conceitos de Programação** ExcelVBABokks.indb 232

15/02/2013 13:59:59



Parte IV

Como se Comunicar

com Seus Usuários

A 5ª Onda

Por Rich Tennant

“Eu experimentei vários programas de planilha,
mas este é o melhor para modelar
colchas de retalho .”

ExcelVBABokks.indb 233

15/02/2013 14:00:00

Nesta parte...

O s cinco capítulos desta parte mostram como
desenvolver caixas de diálogo personalizadas

(também conhecidas como UserForms). Esse recurso

VBA é bastante fácil de usar, depois que você consegue controlar
alguns conceitos básicos. E, se for

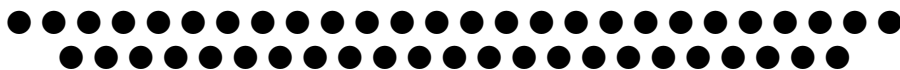
como eu, na verdade, você pode *gostar* de criar caixas de diálogo.

ExcelVBABokks.indb 234

15/02/2013 14:00:00

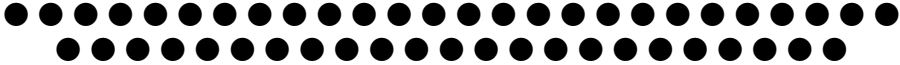
Capítulo 15

Caixas de Diálogo Simples



Neste Capítulo

- ☐ Como poupar tempo usando qualquer das várias alternativas a UserForms
- ☐ Como usar as funções InputBox e MsgBox para obter informações do usuário
- ☐ Como obter do usuário um nome de arquivo e caminho
- ☐ Como obter do usuário um nome de pasta
- ☐ Escrevendo código VBA para executar comandos da faixa de opções (ribbon) que exibem caixas de diálogo integradas ao Excel



Não é possível usar o Excel por muito tempo sem ficar exposto às caixas de diálogo. Elas parecem surgir quase sempre que você

seleciona um comando. O Excel — como a maioria dos programas

Windows — usa caixas de diálogo para obter informações, esclarecer comandos e exibir mensagens. Se você desenvolver macros, pode criar as suas próprias caixas de diálogo que funcionam exatamente como aquelas integradas no Excel. Essas caixas de diálogo personalizadas são chamadas de UserForms (formulários de usuário) em VBA.

Este capítulo não informa nada sobre a criação de UserForms. Ao invés disso, descreve algumas técnicas que podem ser usadas no lugar de UserForms. Entretanto, os Capítulos de 16 a 18 cobrem UserForms.

Alternativas a UserForm

Algumas das macros VBA que você cria se comportam da mesma forma sempre que você as executa. Por exemplo, você pode desenvolver uma macro que forneça uma lista de seus funcionários em uma faixa de planilha. Essa macro sempre produz o mesmo resultado e não requer dados adicionais do usuário.

No entanto, seria possível desenvolver outras macros que se comportem de maneira diferente sob circunstâncias diversas ou que ofereçam opções ao usuário. Em tais casos, a macro pode ser aperfeiçoada com uma caixa de diálogo personalizada. Uma caixa de diálogo personalizada oferece uma maneira simples de obter informações do usuário.

Depois, a sua macro usa tais informações para determinar o que fazer.

UserForms pode ser bem útil, mas criá-las requer tempo. Antes de eu entrar no assunto da criação de UserForms, no próximo capítulo, você precisa conhecer algumas alternativas que, potencialmente, poupam tempo.

ExcelVBABokks.indb 235

15/02/2013 14:00:00

236 Parte IV: Como se Comunicar com Seus Usuários O VBA permite que você exiba vários tipos diferentes de caixas de diálogo

que, às vezes, podem ser usadas no lugar de uma UserForm. É

possível personalizar de alguma maneira essas caixas de diálogo integradas, mas, certamente, elas não oferecem as opções disponíveis em uma UserForm. Porém, em alguns casos, elas são exatamente o que o médico recomendou.

Neste capítulo, você vai ler sobre

5 A função MsgBox

5 A função InputBox

5 O método GetOpenFilename

5 O método GetSaveAsFilename

5 O método FileDialog

Eu também descrevo como usar VBA para exibir as caixas de diálogo integradas do Excel — as caixas de diálogo que o Excel usa para obter informações suas.

A Função MsgBox

Provavelmente você já está familiarizado com a função MsgBox do VBA

— eu a uso bastante nos exemplos deste livro. A função MsgBox, a qual aceita os argumentos mostrados na Tabela 15-1, é útil para exibir informações e obter entrada simples do usuário. Uma função, como você deve lembrar, retorna um valor. No caso da função MsgBox, ela usa uma caixa de diálogo para obter o valor que ela retornar. Continue a ler para ver exatamente como ela funciona.

Eis uma versão simplificada da sintaxe para a função MsgBox:

MsgBox(aviso[, botões][, título])

Tabela 15-1

Argumentos da Função MsgBox

Argumento

O Que Ele Faz

aviso (prompt)

O texto que o Excel exibe na caixa de mensagem

botões (buttons)

Um número que especifica quais botões (e qual ícone)

aparece na caixa de mensagem (opcional)

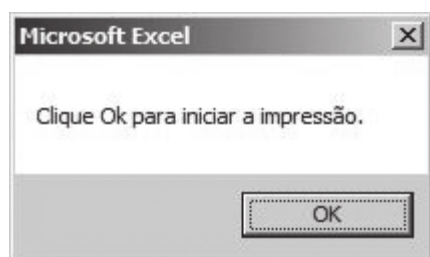
título (title)

O texto que aparece na barra de título da caixa de mensa-

gem (opcional), exibindo uma simples caixa de mensagem

ExcelVBABokks.indb 236

15/02/2013 14:00:00



Capítulo 15: Caixas de Diálogo Simples

Você pode usar a função MsgBox de duas maneiras:

5 Apenas para mostrar uma mensagem ao usuário. Nesse caso, você não se preocupa com o resultado retornado pela função.

5 Para obter uma resposta do usuário. Nesse caso, você se preocupa com o resultado retornado pela função. O resultado

depende do botão que o usuário clicar.

Se você usar a função MsgBox sozinha, não inclua parênteses em torno dos argumentos. O exemplo a seguir exibe apenas uma mensagem e não retorna um resultado. Quando a mensagem é exibida, o código é interrompido até o usuário clicar OK.

```
Sub MsgBoxDemo()
```

```
MsgBox “Clique OK para iniciar a impressão.”
```

```
Sheets(“Results”).PrintOut
```

```
End Sub
```

A Figura 15-1 mostra como essa caixa de mensagem se parece.

Figura 15-1:

Uma

simples

caixa de

mensagem.

Obtendo uma resposta de

uma caixa de mensagem

Se você exibir uma caixa de mensagem que tenha mais do que apenas um botão OK, provavelmente irá querer saber qual botão o usuário clicará. A função MsgBox pode retornar um valor que representa qual botão foi clicado. Você pode designar o resultado da função MsgBox a

uma variável.

No seguinte código, eu uso algumas constantes integradas (as quais descrevo mais adiante, na Tabela 15-2), que facilitam trabalhar com os valores retornados por MsgBox:

```
Sub GetAnswer()
```

```
Dim Ans As Integer
```

```
Ans = MsgBox("Você ainda não almoçou?", vbYesNo)
```

```
Select Case Ans
```

```
Case vbYes
```

```
‘
```

```
... [code if Ans is Yes]...
```

```
Case vbNo
```

```
‘
```

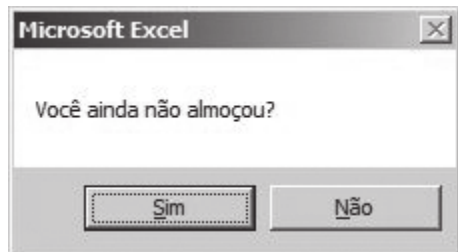
```
...[code if Ans is No]...
```

```
End Select
```

```
End Sub
```

ExcelVBABokks.indb 237

15/02/2013 14:00:01



238 Parte IV: Como se Comunicar com Seus Usuários

A Figura 15-2 mostra como ela se parece. Quando você executa esse procedimento, a variável `Ans` é designada a um valor, seja `vbYes` ou `vbNo`, dependendo de qual botão o usuário clicar. A declaração `Select Case` usa o valor `Ans` para determinar qual ação o código deve executar.

Figura

15-2: Uma

simples

caixa de

mensagem, com

dois

botões.

Você também pode usar o resultado da função `MsgBox` sem usar uma variável, como demonstra o seguinte exemplo:

```
Sub GetAnswer2()
```

```
    If MsgBox("Continue?", vbYesNo) = vbYes Then
```

```
        '
```

```
        ...[code if Yes is clicked]...
```

```
    Else
```

```
        '
```

...[code if Yes is not clicked]...

End If

End Sub

Personalizando caixas de mensagem

A flexibilidade dos botões de argumento facilitam a personalização de suas caixas de mensagem. Você pode especificar quais botões exibir, determinar se um ícone aparece e decidir qual botão é o padrão (o botão padrão é “clicado” se o usuário pressionar Enter).

A Tabela 15-2 relaciona algumas das constantes integradas que você pode usar para os botões de argumento. Se preferir, você pode usar o valor ao invés de uma constante (mas eu creio que usar as constantes integradas é muito mais fácil).

Tabela 15-2

Constantes Usadas na Função MsgBox

Constante

Valor

O Que Ela Faz

vbOKOnly

0

Exibe apenas o botão OK.

vbOKCancel

1

Exibe os botões OK e Cancel.

vbAbortRetryIgnore 2

Exibe os botões Abort, Retry e Ignore

vbYesNoCancel

3

Exibe os botões Yes, No e Cancel.

vbYesNo

4

Exibe os botões Yes e No.

ExcelVBABokks.indb 238

15/02/2013 14:00:01

Capítulo 15: Caixas de Diálogo Simples

239

Constante

Valor O Que Ela Faz

vbRetryCancel

5

Exibe os botões Retry e Cancel.

vbCritical 16

Exibe o ícone Mensagem Crítica.

vbQuestion

32

Exibe o ícone Consulta de Aviso.

vbExclamation

48

Exibe o ícone Mensagem de Aviso.

vbInformation

Exibe o ícone Mensagem de Informação.

`vbDefaultButton1 0`

O botão padrão é o primeiro.

`vbDefaultButton2 256`

O botão padrão é o segundo.

`vbDefaultButton3`

512

O botão padrão é o terceiro.

`vbDefaultButton4`

768

O botão padrão é o quarto.

Para usar mais de uma destas constantes como um argumento, basta conectá-las a um operador `+`. Por exemplo, para exibir uma caixa de mensagem com Sim e Não e um ícone de ponto de exclamação, use a seguinte expressão como o segundo argumento de `MsgBox`: `vbYesNo + vbExclamation`

Ou, se você preferir tornar o seu código menos compreensível, use um valor de 52 (isto é, `4 + 48`).

O exemplo a seguir usa uma combinação de constantes para exibir uma caixa de mensagem com um botão Sim e um botão Não (`vbYesNo`),

assim como um ícone de ponto de interrogação (`vbQuestion`). A

constante `vbDefaultButton2` designa o segundo botão (Não) como o botão padrão – isto é, o botão que é clicado se o usuário pressionar Enter. Para simplificar, eu atribuí essas constantes à variável `Config` e depois, usei `Config` como o segundo argumento na função `MsgBox`.

`Sub GetAnswer3()`

Dim Config As Integer

Dim Ans As Integer

Config = vbYesNo + vbQuestion + vbDefaultButton2

Ans = MsgBox(“Emitir o relatório mensal?”, Config)

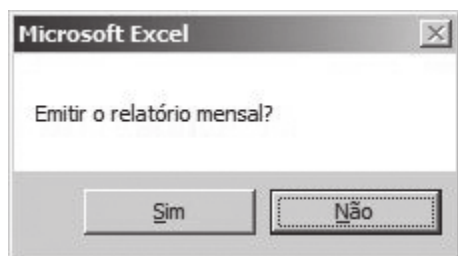
If Ans = vbYes Then RunReport

End Sub

A Figura 15-3 mostra a caixa de mensagem que o Excel exibe quando você executa o procedimento GetAnswer3. Se o usuário clicar o botão Sim, a rotina executa o procedimento chamado RunReport (que não é mostrado). Se o usuário clicar o botão Não (ou pressionar Enter), a rotina termina sem ação. Pelo fato de que eu omiti o argumento título na função MsgBox, o Excel usa o título padrão, Microsoft Excel.

ExcelVBABokks.indb 239

15/02/2013 14:00:02



240 Parte IV: Como se Comunicar com Seus Usuários

Figura 15-3:

Os botões

de argumento da

função

MsgBox

determinam

o que

aparece na

caixa de

mensagem.

A seguinte rotina oferece um outro exemplo do uso da função MsgBox:
Sub GetAnswer4()

Dim Msg As String, Title As String

Dim Config As Integer, Ans As Integer

Msg = “Você deseja emitir o relatório mensal?”

Msg = Msg & vbNewLine & vbNewLine

Msg = Msg & “O relatório mensal será emitido “

Msg = Msg & “em aproximadamente 15 minutos. Serão “

Msg = Msg & “geradas 30 páginas para “

Msg = Msg & “todos os escritórios de vendas do “

Msg = Msg & “mês atual.”

Title = “XYZ Marketing Company”

Config = vbYesNo + vbQuestion

Ans = MsgBox(Msg, Config, Title)

If Ans = vbYes Then RunReport

End Sub

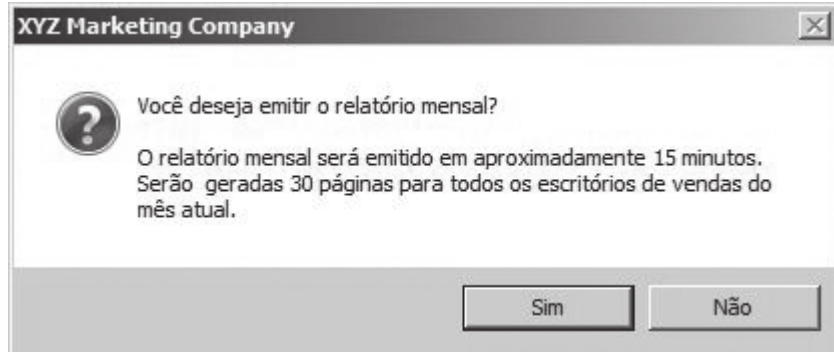
Este exemplo demonstra uma maneira eficiente de especificar uma mensagem mais longa em uma caixa de mensagem. Eu uso uma variável (Msg) e o operador de concatenação (&) para montar a mensagem em uma série de declarações. A constante VBNewLine inicia uma nova linha (use-a duas vezes para inserir uma linha em branco). Eu também uso o argumento title para exibir um título diferente na caixa de mensagem. A Figura 15-4 mostra a caixa de mensagem que o Excel exibe quando você executa esse procedimento.

Exemplos anteriores usaram constantes (tais como vbYes e vbNo) para o valor retornado de uma função MsgBox. Além destas duas constantes, a Tabela 15-3 relaciona algumas outras.

E isso é praticamente tudo o que você precisa saber sobre a função MsgBox. Porém, use caixas de mensagem com cautela. Normalmente, não há motivo para exibir caixas de mensagem sem objetivo. Por exemplo, as pessoas tendem a ficar aborrecidas quando vêm uma caixa de mensagem diariamente, que diz, ‘Bom dia, obrigado por carregar a pasta de trabalho de Projeção Orçamentária’.

ExcelVBABokks.indb 240

15/02/2013 14:00:02



Capítulo 15: Caixas de Diálogo Simples 241

Figura 15-4:

Esta caixa
de diálogo,
exibida pela
função
MsgBox,
exibe um
título, um
ícone e dois
botões,

Tabela 15-3 Constantes Usadas como Valores de Retorno da Função MsgBox

Constante

Valor

O Que Significa

vbOK

1

Usuário clicou OK

vbCancel

2

Usuário clicou Cancelar

vbAbort

3

Usuário clicou Anular

vbRetry

4

Usuário clicou Repetir

vbIgnore

5

Usuário clicou Ignorar

vbYes

6

Usuário clicou Sim

vbNo

7

Usuário clicou Não

A Função InputBox

A função InputBox do VBA é útil para obter parte de uma informação do usuário. Tal informação poderia ser um valor, uma string de texto ou mesmo uma faixa de endereço. Essa é uma boa alternativa para desenvolver uma UserForm, quando você só precisar obter um valor.

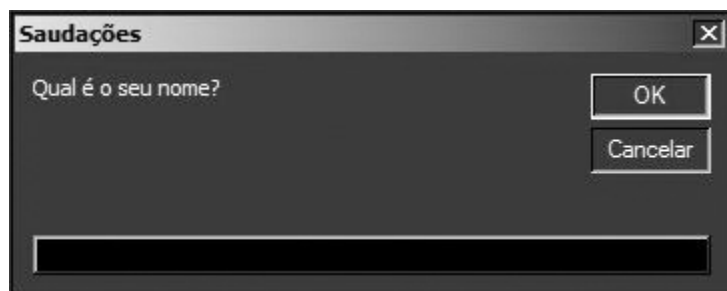
Sintaxe InputBox

Eis uma versão simplificada da sintaxe para a função InputBox:
InputBox(prompt[, title][, default])

A função InputBox aceita os argumentos relacionados na Tabela 15-4.

ExcelVBABokks.indb 241

15/02/2013 14:00:03



242 Parte IV: Como se Comunicar com Seus Usuários

Tabela 15-4

Argumentos da Função InputBox

Argumento

O Que Ele Significa

Aviso (Prompt)

O texto exibido na caixa de entrada

Título (Title)

O texto exibido na barra de título da caixa de entrada
(opcional)

Padrão(Default)

O valor padrão da entrada do usuário (opcional)

Um exemplo de InputBox

Eis um exemplo mostrando como você pode usar a função InputBox:
TheName = InputBox(“Qual é o seu nome?”, “saudações”)

Quando você executa essa declaração VBA, o Excel exibe a caixa de diálogo mostrada na Figura 15-5. Note que esse exemplo só usa os dois primeiros argumentos e não fornece um valor padrão. Quando o usuário entra com um valor e clica OK, a rotina atribui o valor à variável TheName.

Figura 15-5:

A função

InputBox

exibe esta

caixa de

diálogo.

O seguinte exemplo usa o terceiro argumento e oferece um valor padrão. O valor padrão é o nome de usuário armazenado pelo Excel (a propriedade UserName do objeto Application).

```
Sub GetName()
```

```
Dim DefName As String
```

```
Dim TheName As String
```

```
DefName = Application.UserName
```

```
TheName = InputBox("Qual é o seu nome?", _
```

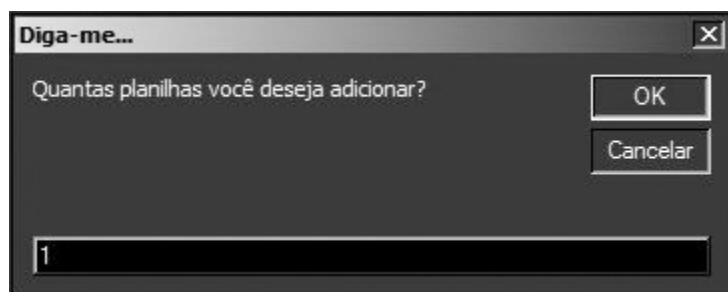
```
"Saudações", DefName)
```

```
End Sub
```


A caixa de entrada sempre exibe um botão Cancelar. Se o usuário clicar em Cancelar, a função InputBox retorna uma string vazia.

ExcelVBABokks.indb 242

15/02/2013 14:00:03



Capítulo 15: Caixas de Diálogo Simples

243

A função InputBox do VBA sempre retorna uma string, portanto, se **BRE-SE**

você precisa obter um valor, o seu código precisará fazer alguma **LEM**

verificação adicional. O seguinte exemplo usa a função InputBox para obter um número. Ele usa a função IsNumeric (É numérico) para verificar se a string é um número. Se a string contiver um número, está tudo bem. Se a entrada do usuário não puder interpretar como um número, o código exibe uma caixa de mensagem.

```
Sub GetName2AddSheet()
```

```
Dim Prompt As String
```

```
Dim Caption As String
```

```
Dim DefValue As Integer
```

Dim NumSheets As String

Prompt = “Quantas planilhas você deseja adicionar?”

Caption = “Diga-me...”

DefValue = 1

NumSheets = InputBox(Prompt, Caption, DefValue)

If NumSheets = “ “ Then Exit Sub ‘Canceled

If IsNumeric(NumSheets) Then

If NumSheets > 0 Then Sheets.Add _

Count: = NumSheets

Else

MsgBox “Número inválido”

End If

End Sub

A Figura 15-6 mostra a caixa de diálogo que esta rotina produz.

Figura 15-6:

Um outro

exemplo de

uso da

função

InputBox.

As informações apresentadas nesta seção aplicam-se à função InputBox do VBA. Além disso, você tem acesso ao método InputBox, que é um **DE ESPECIALISTA**

O P

método do objeto Application.

A P

Uma grande vantagem de usar o método InputBox é que seu código pode solicitar a seleção de uma faixa. Depois, o usuário pode selecionar a faixa na planilha destacando as células. Eis um rápido exemplo que solicita o usuário a selecionar uma faixa:

ExcelVBABokks.indb 243

15/02/2013 14:00:04

244 Parte IV: Como se Comunicar com Seus Usuários Sub
GetRange()

Dim Rng As Range

On Error Resume Next

Set Rng = Application.InputBox _

(prompt: = "Especifique uma faixa:", Type: = 8)

If Rng Is Nothing Then Exit Sub

MsgBox “Você selecionou a faixa “ & Rng.Address

End Sub

Neste simples exemplo, o código diz ao usuário o endereço da faixa que foi selecionada. Na vida real, o seu código poderia, de fato, fazer algo útil com a faixa selecionada.

O método Application.InputBox é semelhante à função InputBox do VBA, mas também apresenta algumas diferenças. Para detalhes completos, clique o sistema de Ajuda.

O Método GetOpenFilename

Se o seu procedimento VBA precisar pedir ao usuário um nome de arquivo, você *poderia* usar a função InputBox. Geralmente, uma caixa de entrada não é a melhor ferramenta para esse trabalho, pelo fato de a maioria dos usuários achar difícil lembrar caminhos, barras invertidas, nomes de arquivos e extensões de arquivo. Em outras palavras, é muito mais fácil cometer um erro de digitação ao digitar um nome de arquivo.

Para uma melhor solução desse problema, use o método GetOpenFilename do objeto Application, o qual garante que o seu código ofereça um nome de arquivo válido, incluindo o seu caminho completo. O

método GetOpenFilename exibe a conhecida caixa de diálogo Abrir (igual à caixa de diálogo que o Excel exibe quando você escolhe Arquivo⇒Abrir.

Na verdade, o método GetOpenFilename não abre o arquivo especifica-BRE-SE

do. Esse método apenas retorna o nome de arquivo selecionado pelo **LEM**

usuário como uma string. Depois, você pode escrever um código para o que quiser com o nome de arquivo.

A sintaxe para o método GetOpenFilename

A sintaxe oficial para o método GetOpenFilename é como a seguir:
object.GetOpenFilename ([fileFilter], [filterIndex],

[title],[buttonText], [multiSelect])

ExcelVBABokks.indb 244

15/02/2013 14:00:04

Capítulo 15: Caixas de Diálogo Simples 245

O método GetOpenFilename toma os argumentos opcionais mostrados na Tabela 15-5.

Tabela 15-5 Argumentos do Método GetOpenFilename

Argumento

O que ele faz

FileFilter

Determina os tipos de arquivos que aparecem na caixa de

diálogo (por exemplo, *.TXT). Você pode especificar vários

filtros diferentes a partir dos quais o usuário pode escolher.

FilterIndex

Determina qual dos filtros a caixa de diálogo exibe por

padrão.

Title

Especifica a legenda para a barra de título da caixa de diálogo.

ButtonText

Ignorado (usado apenas para a versão Macintosh do Excel).

MultiSelect

Se True, o usuário pode selecionar múltiplos arquivos.

Um exemplo de GetOpenFilename

O argumento FileFilter determina o que aparece na lista drop-down quanto aos tipos de arquivo na caixa de diálogo. Esse argumento consiste de pares de strings de filtro de arquivo seguidos pela especificação de filtro de arquivo curinga, com vírgulas separando cada parte.

Se omitido, esse argumento padroniza para o seguinte:

All Files (.), .

Observe que esta string consiste de duas partes:

All Files (.)

e

.

A primeira parte desta string consiste do texto exibido na lista drop-down de tipos de arquivos. A segunda parte determina quais arquivos a caixa de diálogo exibe. Por exemplo, . significa *todos os arquivos*.

O código no exemplo a seguir adianta uma caixa de diálogo que pede um nome de arquivo ao usuário. O procedimento define cinco filtros de arquivo. Veja que eu uso a sequência de continuação de linha do VBA para configurar a variável Filter; fazer isso ajuda a simplificar esse bem complicado argumento.

ExcelVBABokks.indb 245

15/02/2013 14:00:05

246 Parte IV: Como se Comunicar com Seus Usuários Sub
GetImportFileName ()

Dim Finfo As String

Dim FilterIndex As Integer

Dim Title As String

Dim FileName As Variant

‘ Set up list of file filters

Finfo = “Text Files (*.txt),*.txt,” & _

“Lotus Files (*.prn),*.prn,” & _

“Comma Separated Files (*.csv),*.csv,” & _

“ASCII Files (*.asc),*.asc,” & _

“All Files (.),.”

‘ Display . by default

FilterIndex = 5

‘ Set the dialog box caption

Title = “Selecione um arquivo para importar”

‘ Get the filename

FileName = Application.GetOpenFilename (Finfo, _

FilterIndex, Title)

‘ Handle return info from dialog box

If FileName = False Then

MsgBox “Nenhum arquivo foi selecionado.”

Else

MsgBox “Você selecionou “ & FileName

End If

End Sub

A Figura 15-7 mostra a caixa de diálogo que o Excel exibe quando você executa este procedimento. Em um aplicativo real, você poderia fazer algo mais significativo com o nome de arquivo. Por exemplo, você poderia querer abri-lo usando uma declaração como esta:
Workbooks.Open FileName

Observe que a variável FileName é declarada como um tipo de dados Variant. Se o usuário clicar em Cancelar, essa variável armazena um valor **DE ESPECIALISTA**

O

Booleano (Falso). Caso contrário, FileName é uma string. Portanto,

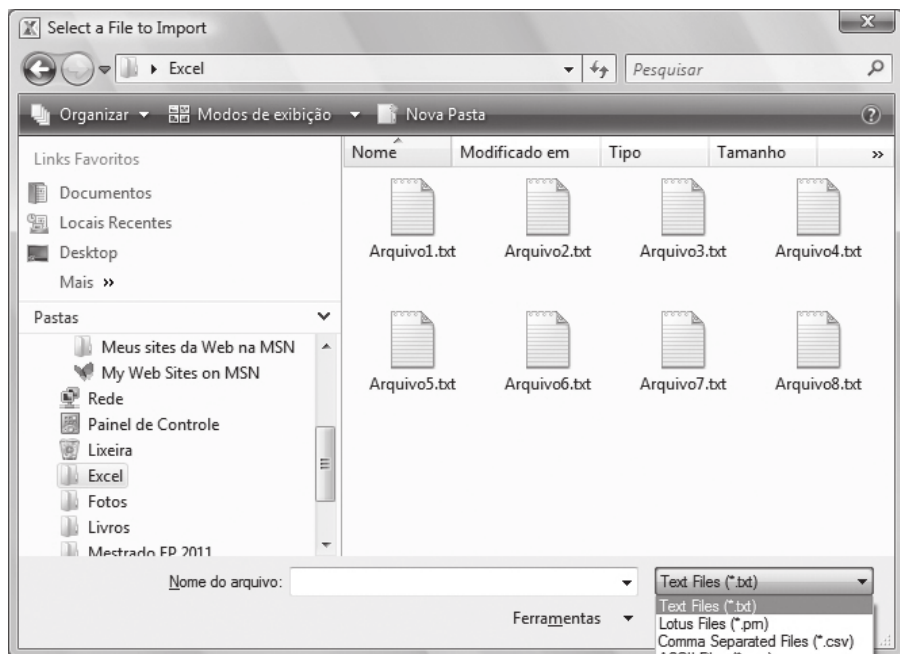
usar **P A P**

um tipo de dados Variant lida com ambas as possibilidades.

A propósito, a caixa de diálogo pode parecer diferente, dependendo de qual versão de Windows você usa.

ExcelVBABokks.indb 246

15/02/2013 14:00:05



Capítulo 15: Caixas de Diálogo Simples

247

Figura 15-7:

O método

GetOpenFilename

exibe uma

caixa de

diálogo

personalizada e retorna

o caminho e

o nome do

arquivo

selecionado.

Ele não abre

o arquivo.

Selecionando múltiplos arquivos

Se o argumento MultiSelect para o método GetOpenFilename for True, o usuário pode selecionar múltiplos arquivos na caixa de diálogo, pressionando Ctrl enquanto clica nos arquivos. Nesse caso, o método GetOpenFilename retorna um array de nomes de arquivos. O seu código deve fazer um loop através do array para identificar cada nome de arquivo selecionado, como demonstra o seguinte exemplo: Sub GetImportFileName2()

```
Dim FileNames As Variant
```

```
Dim Msg As String
```

```
Dim I As Integer
```

```
FileNames = Application.GetOpenFilename _
```

```
(MultiSelect:=True)
```

```
If IsArray(FileNames) Then
```

‘ Mostrar o caminho completo e o nome dos arquivos

Msg = “Você selecionou:” & vbNewLine

For I = LBound(FileNames) To _

Ubound(FileNames)

Msg = Msg & FileNames(i) & vbNewLine

Next i

MsgBox Msg

Else

‘ Botão de cancelar clicado

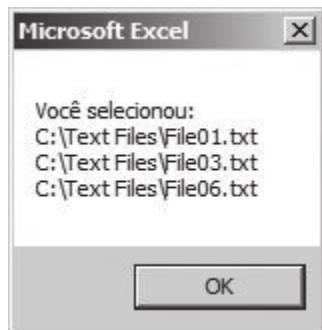
MsgBox “Nenhum arquivo foi selecionado.”

End If

End Sub

ExcelVBABokks.indb 247

15/02/2013 14:00:06



248 Parte IV: Como se Comunicar com Seus Usuários

A Figura 15-8 mostra o resultado ao rodar este procedimento. A caixa de mensagem exibe os nomes de arquivo que foram selecionados:

Figura 15-8:

Selecione

múltiplos

nomes de

arquivos

usando o

método

GetOpenFilename.

Observe que usei um argumento nomeado para o método GetOpenFilename. Também, configurei o argumento MultiSelect (Seleção múltipla) para True. Os outros argumentos são omitidos, portanto eles tomam seus valores padrão. Usar argumentos nomeados elimina a necessidade de especificar argumentos que não são usados.

A variável FileNames é definida como um tipo de dados Variant. Eu uso a função IsArray para determinar se FileName contém um array. Se **DE ESPECIALISTA**

O

assim for, o código usa as funções Lbound e UBound do VBA para deter-P A P

minar os limites mais baixos e mais altos do array e monta uma

mensagem que consiste de cada elemento de array. Se a variável FileNames não for um array, significa que o usuário clicou no botão Cancelar (a variável FileNames contém um array, mesmo que apenas um arquivo seja selecionado).

O Método GetSaveAsFileName

O método GetSaveAsFilename do Excel funciona exatamente como o método GetOpenFilename, mas ele exibe a caixa de diálogo Salvar Como do Excel, ao invés de sua caixa de diálogo Abrir. O método GetSaveAsFilename obtém um caminho e nome de arquivo do usuário, mas não faz nada com isso. É você que escreve o código que, de fato, salva o arquivo.

A sintaxe para esse método é apresentada a seguir:

```
object.GetSaveAsFilename ([InitialFilename], [FileFilter],
```

```
[FilterIndex], [title], [ButtonText])
```

O método GetSaveAsFilename toma os argumentos da Tabela 15-6, todos os quais são opcionais.

ExcelVBABokks.indb 248

15/02/2013 14:00:06

Capítulo 15: Caixas de Diálogo Simples

249

Tabela 15-6 Argumentos do Método GetSaveAsFilename

Argumento

O Que Ele Faz

InitialFileName

Especifica um nome de arquivo padrão que aparece na caixa

(de diálogo) FileName.

FileFilter

Determina os tipos de arquivos que o Excel exibe na caixa de

diálogo (por exemplo, *.TXT). Você pode especificar vários

filtros diferentes a partir dos quais o usuário pode escolher.

FilterIndex

Determina qual dos filtros de arquivo o Excel exibe por padrão.

Title

Define uma legenda para a barra de título da caixa de diálogo.

Como Obter um Nome de Pasta

Às vezes, você não precisa obter um nome de arquivo, só precisa de um nome de pasta. Se esse for o caso, o objeto FileDialog é exatamente o que o médico recomendou.

O seguinte procedimento exibe uma caixa de diálogo que permite ao usuário selecionar um diretório. O nome do diretório selecionado (ou

“Cancelado”) é então exibido, usando a função MsgBox.

Sub GetAFolder()

With Application.FileDialog(msoFileDialogFolderPicker)

.InitialFileName = Application.DefaultFilePath & “\”

.Title = “Please select a location for the backup”

.Show

If .SelectedItems.Count = 0 then

MsgBox “Cancelado”

Else

MsgBox .SelectedItems(1)

End If

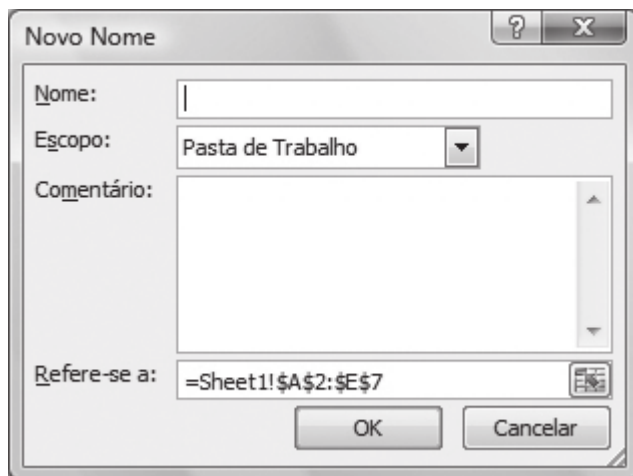
End With

End Sub

O objeto FileDialog permite que você especifique o diretório de início, especificando um valor para a propriedade InitialFileName. Neste caso, o código usa o caminho de arquivo padrão do Excel como o diretório inicial.

ExcelVBABokks.indb 249

15/02/2013 14:00:07



250 Parte IV: Como se Comunicar com Seus Usuários

Exibindo as Caixas de Diálogo

Integradas do Excel

Uma maneira de ver um VBA é que ele é uma ferramenta que permite imitar os comandos do Excel. Por exemplo, veja esta declaração VBA: `Range("A1:A12").Name = "MonthNames"`

Executar essa declaração VBA tem o mesmo efeito que escolher

Fórmulas e Nomes Definidos e Definir Nome para exibir a caixa de

diálogo Novo Nome, e depois, digitar MonthNames na caixa Nome e A1:A12 na caixa Refere-se a clicar OK.

Quando você executa a declaração VBA, a caixa de diálogo Novo Nome não aparece. Quase sempre é isso que você quer que aconteça: você não quer caixas de diálogo passando na tela enquanto a sua macro executa.

No entanto, em alguns casos você pode querer que o seu código exiba uma das caixas de diálogo integradas do Excel e permita ao usuário fazer escolhas. Isso pode ser feito usando VBA para executar um comando da faixa de opções. Eis um exemplo que exibe a caixa de diálogo Novo Nome (veja a Figura 15-9).

`Application.CommandBars.ExecuteMso ("NameDefine")`

Figura 15-9:

Exibindo

uma das

caixas de

diálogo do

Excel

usando VBA.

O seu código VBA não pode obter quaisquer informações da caixa de **BRE-SE**

diálogo. Por exemplo, se você executar o código para exibir a caixa de **LEM**

diálogo Novo Nome, o seu código não pode obter o nome fornecido pelo usuário, ou a faixa em que ele foi nomeado.

O `ExecuteMso` é um método do objeto `CommandBars` e aceita um

argumento, um parâmetro `idMso`, que representa um controle da faixa de opções. Infelizmente, esses parâmetros não estão relacionados no `ExcelVBABokks.indb` 250

15/02/2013 14:00:07

Capítulo 15: Caixas de Diálogo Simples

251

sistema de ajuda. O código que usa o método `ExecuteMso` não é compatível com versões anteriores ao Excel 2007.

Você pode fazer o download de um arquivo a partir do Web site deste livro, que relaciona todos os nomes de parâmetro de comando da faixa de opções do Excel.

Eis um outro exemplo de usar o método `ExecuteMso`. Esta declaração, quando executada, exibe a guia Fonte da caixa de diálogo Formatar Células:

```
Application.CommandBars.ExecuteMso("FormatCellsFontDialog")
```

 Se

você tentar exibir uma caixa de diálogo integrada em um contexto errado, o Excel exibe uma mensagem de erro. Por exemplo, eis uma declara-

ção que exibe a caixa de diálogo Formatar Número:

`Application.commandBars.ExecuteMso ("NumberFormatsDialog")`

Se você executar esta declaração quando ela não for apropriada (por exemplo, uma Shape estiver selecionada), o Excel exibe uma mensagem de erro, pois aquela caixa de diálogo só é adequada às células de planilha.

ExcelVBABokks.indb 251

15/02/2013 14:00:08

252 Parte IV: Como se Comunicar com Seus Usuários

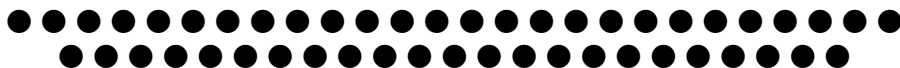
ExcelVBABokks.indb 252

15/02/2013 14:00:08

Capítulo 16

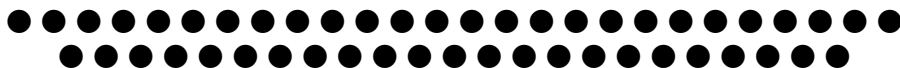
Princípios Básicos

de UserForm



Neste Capítulo

- ☐ Descobrindo quando usar UserForms
- ☐ Como entender objetos UserForm
- ☐ Exibindo UserForm
- ☐ Como criar um Userform que funcione com uma macro útil



Um UserForm é útil se a sua macro VBA precisar obter informações de um usuário. Por exemplo, a sua macro pode ter algumas

opções que podem ser especificadas em um UserForm. Se forem necessárias apenas algumas partes de informações (por exemplo, uma resposta Sim/Não ou uma string de texto), uma das técnicas descritas no Capítulo 15 pode fazer o trabalho. Porém, se você precisar de mais informações, deve criar um UserForm. Neste capítulo, eu o apresento aos UserForms. Você ficará feliz em conhecê-los.

Como Saber Quando Usar um UserForm

Esta seção descreve uma situação onde um UserForm é útil. A seguinte macro muda o texto em cada célula na seleção para letras maiúsculas.

Isso é feito usando a função UCase (Uppercase – Letras maiúsculas) integrada do VBA.

```
Sub ChangeCase ()
```

```
Dim WorkRange As Range
```

```
‘ Sai se a faixa não foi selecionada
```

```
If TypeName(Selection) <> “Range” Then Exit Sub
```

```
‘ Processa apenas células de texto, sem fórmulas
```

```
On error Resume Next
```

```
Set WorkRange = Selection.SpecialCells _
```

```
(xlCellTypeConstants, xlCellTypeConstants)
```

```
For Each cell In WorkRange
```

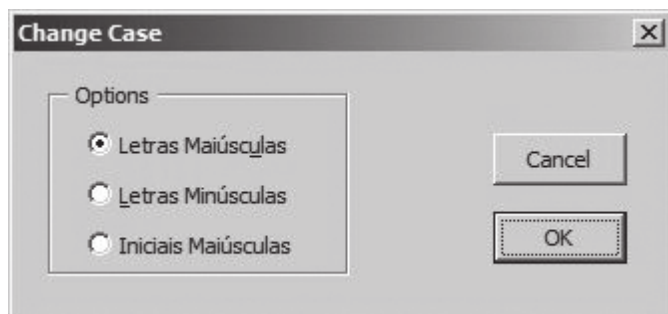
```
cell.Value = Ucase(cell.Value)
```

Next cell

End Sub

ExcelVBABokks.indb 253

15/02/2013 14:00:08



254 Parte IV: Como se Comunicar com Seus Usuários

Você pode tornar essa macro ainda mais útil. Por exemplo, seria interessante se a macro também pudesse alterar o texto nas células para minúsculas ou para o tipo apropriado (colocando em maiúscula a primeira letra de cada palavra). Uma abordagem é criar duas macros adicionais — uma para letras minúsculas e uma para maiúsculas. Outra abordagem é modificar a macro para lidar com outras opções. Se você usar a segunda abordagem, precisa de algum método para perguntar ao usuário qual tipo de alteração fazer às células.

A solução é exibir uma caixa de diálogo como a mostrada na Figura 16-1. Você cria esta caixa de diálogo em um UserForm no VBE e a exibe, usando uma macro VBA. Na próxima seção, ofereço instruções passo a passo para criar esta caixa de diálogo. Antes de me aprofundar, preparo o palco com algum material introdutório.

Figura 16-1:

Você pode

obter

informações

do usuário,
exibindo um
UserForm.

Em VBA, o nome oficial de uma caixa de diálogo é UserForm. Porém, uma UserForm é, na verdade, um objeto que contém o que normalmente é **DE ESPECIALISTA**

O P

conhecido como uma *caixa de diálogo*. Esta distinção não é importante,
A P

portanto, eu costumo usar esses termos alternadamente.

Criando UserForms: Uma Visão Geral

Para criar um UserForm, normalmente você segue as seguintes etapas gerais:

1. Determine como a caixa de diálogo será usada e onde ela será exibida em sua macro VBA.

2. Pressione Alt + F11 para ativar o VBE e insira um novo objeto UserForm.

Um objeto UserForm contém um único UserForm.

3. Acrescente controles ao UserForm.

Os controles incluem itens tais como caixas de texto, botões, caixas de verificação e caixas de listas.

ExcelVBABokks.indb 254

15/02/2013 14:00:09

Capítulo 16: Princípios Básicos de UserForm 255

4. Use a janela Propriedades para modificar as propriedades para os controles ou para o próprio UserForm.

5. Escreva procedimentos que lidam com eventos para os

controles (por exemplo, uma macro que execute quando o usuário

clicar um botão na caixa de diálogo).

Esses procedimentos são armazenados na janela de código para o objeto UserForm.

6. Escreva um procedimento (armazenado em um módulo VBA) que exiba a caixa de diálogo ao usuário.

Não se preocupe se algumas dessas etapas parecem estranhas. Eu dou mais detalhes nas seções seguintes, juntamente com instruções passo a passo para criar um UserForm.

Quando você está projetando um UserForm, está criando o que os desenvolvedores chamam de Graphical User Interface (GUI – Interface Gráfica de Usuário) em seu aplicativo. Use algum tempo para pensar em como o seu formulário deve se parecer e como os seus usuários podem querer interagir com os elementos no UserForm. Tente guiá-los através das etapas que eles precisam tomar no formulário, considerando cuidadosamente a organização e as palavras dos controles. Como a maioria das coisas relacionadas ao VBA, quanto mais você fizer, mais fácil será.

Trabalhando com UserForms

Cada caixa de diálogo que você cria é armazenada em seu próprio objeto UserForm — uma caixa de diálogo UserForm. Você cria e acessa esses UserForms no Visual Basic Editor.

Inserindo um novo UserForm

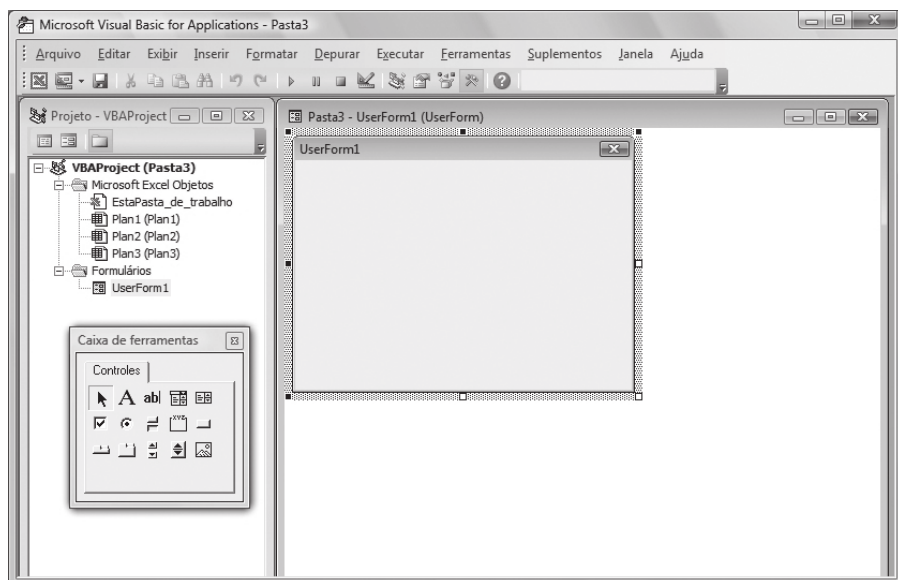
Insira um novo objeto UserForm com as seguintes etapas:

- 1. Ative o VBE, pressionando Alt + F11.**
- 2. Selecione a pasta de trabalho na janela de projeto.**
- 3. Escolha Inserir⇒UserForm.**

O VBE insere um novo objeto UserForm, o qual contém uma caixa de diálogo vazia.

A Figura 16-2 exibe um UserForm — uma caixa de diálogo vazia, precisando de alguns controles.

15/02/2013 14:00:09



256 Parte IV: Como se Comunicar com Seus Usuários

Figura 16-2:

Um novo
objeto
UserForm.

Adicionando controles a um UserForm

Quando você ativa um UserForm, o VBE exibe a Caixa de ferramentas em uma janela flutuante, conforme mostrado na Figura 16-2. Você usa as ferramentas na Caixa de ferramentas para acrescentar controles ao seu UserForm. Se a Caixa de ferramentas não aparecer quando o seu UserForm for ativado, escolha **Exibir** > **Caixa de ferramentas**.

Para acrescentar um controle, basta clicar o controle desejado na Caixa de ferramentas e arrastá-lo para a caixa de diálogo, para criar o controle. Depois de ter adicionado um controle, você pode movê-lo e redimensioná-lo, usando as técnicas padrão.

A Tabela 16-1 indica as várias ferramentas, assim como suas funções.

Para determinar qual é cada ferramenta, passe o cursor do seu mouse sobre o controle e leia a pequena descrição pop-up.

Tabela 16-1

Controles de Caixa de Ferramentas

Controle

O Que Ele Faz

Rótulo

Exibe texto.

Caixa de texto

Permite ao usuário inserir texto.

Caixa combinações

Exibe uma listagem drop-down

Caixa de listagem

Exibe uma lista de itens.

Caixa de seleção

Útil para opções on/off ou sim/não.

Botão de opção

Usado em grupos; permite ao usuário selecionar uma

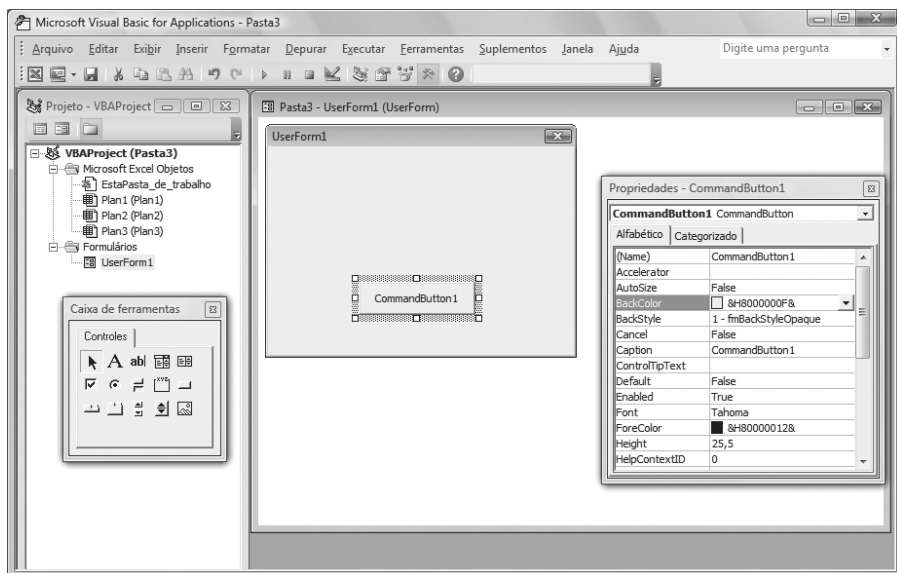
entre várias opções.

Botão de ativação

Um botão que está ativado (on) ou desativado (off)

Quadro

Um contêiner para outros controles.



Capítulo 16: Princípios Básicos de UserForm 257

Controle

O que ele faz

Botão de comando

Um botão clicável

TabStrip

Exibe tabs guias.

Multipágina

Um contêiner tabulado para outros objetos.

Barra de rolagem

Uma barra que pode ser rolada.

Botão de rotação

Um botão clicável usado com frequência para muda

um valor).

Imagem

Contém uma imagem.

RefEdit

Permite ao usuário selecionar uma faixa.

Mudando propriedades em um controle UserForm

Cada controle que você acrescenta a um UserForm tem uma quantidade de propriedades que determina como o controle se parece e se comporta. Além disso, o próprio UserForm tem o seu conjunto de propriedades. Você pode mudar essas propriedades com a janela Propriedades. A Figura 16-3 mostra a janela de propriedades quando um controle Botão de comandos é selecionado.

A janela Propriedades aparece quando você pressiona F4, e as propriedades mostradas nessa janela dependem do que está selecionado. Se você selecionar um controle diferente, as propriedades mudam para aqueles apropriados àquele controle. Para ocultar a janela Propriedades, clique no botão Fechar em sua barra de título.

Figura

16-3: Use

as janelas

Propriedades para

mudar as

propriedades dos

controles

de

UserForm.

258 Parte IV: Como se Comunicar com Seus Usuários As propriedades para os controles incluem o seguinte:

5 Nome

5 Largura

5 Altura

5 Valor

5 Legenda

Cada controle tem o seu próprio conjunto de propriedades (embora muitos controles tenham algumas propriedades em comum). Para mudar uma propriedade usando a janela Propriedades:

- 1. Assegure-se de que o controle certo esteja selecionado no UserForm.**
- 2. Assegure-se de que a janela Propriedades esteja visível (se não estiver, pressione F4).**
- 3. Na janela Propriedades, clique na propriedade que deseja alterar.**
- 4. Faça a alteração na parte certa da janela Propriedades.**

Se você selecionar o próprio UserForm (não um controle no UserForm), pode usar a janela Propriedades para ajustar as propriedades do UserForm.

O Capítulo 17 informa tudo o que você precisa saber sobre trabalhar com controles de caixa de diálogo.

Algumas das propriedades de UserForm servem como configurações
DICA

padrão para os novos controles que você arrasta para dentro do UserForm. Por exemplo, se você mudar a propriedade Font em um UserForm, os controles que você acrescentar usarão aquela mesma fonte. Controles que já estão no UserForm não são alterados.

Observando a janela de Código de UserForm

Cada objeto UserForm tem um módulo de código que contém o código VBA (os procedimentos que lidam com eventos) que é executado

quando o usuário trabalhar com a caixa de diálogo. Para ver o módulo de código, pressione F7. A janela de código fica vazia até que você adicione alguns procedimentos. Pressione Shift + F7 para retornar à caixa de diálogo.

Eis outra maneira de alternar entre a janela de código e a exibição de UserForm: Use os botões Exibir Código e Exibir Objeto na barra de

título da janela de projeto. Ou clique com o botão direito no UserForm e escolha View Code. Se você estiver vendo o código, clique duas vezes no nome UserForm na janela de projeto para voltar ao UserForm.

ExcelVBABokks.indb 258

15/02/2013 14:00:09

Capítulo 16: Princípios Básicos de UserForm 259

Exibindo um UserForm

Você exibe um UserForm usando o método Show do UserForm em um procedimento VBA.

A macro que exibe a caixa de diálogo deve estar em um módulo VBA
BRE-SE

— não na janela de código no UserForm.

LEM

O seguinte procedimento exibe a caixa de diálogo chamada
UserForm1: Sub ShowDialog()

UserForm1.Show

‘ Outras declarações entram aqui

End Sub

Quando o Excel exibe a caixa de diálogo, a macro ShowDialog é interrompida, até o usuário fechar a caixa de diálogo. Depois, o VBA executa quaisquer declarações restantes no procedimento. Na maior parte do tempo, você não tem mais código no procedimento. Como verá mais adiante, você coloca os seus procedimentos de lidar com eventos na janela de código para o UserForm.

Usando informações de um UserForm

O VBE oferece um nome para cada controle que você acrescenta a um UserForm. O nome do controle corresponde à sua propriedade Name.

Use esse nome para se referir a um controle em especial em seu

código.

Por exemplo, se você acrescentar um controle CheckBox (Caixa de Verificação) a um UserForm nomeado como UserForm1, por padrão, o controle CheckBox é nomeado CheckBox1. A seguinte declaração faz esse controle aparecer com uma marca de verificação:

```
UserForm1.CheckBox1.Value = True
```

Na maior parte do tempo, você escreve o código para um UserForm no módulo de código do UserForm. Se esse for o caso, você pode omitir o objeto qualificador de UserForm e escrever a declaração assim:

```
CheckBox1.Value = True
```

O seu código VBA também pode verificar várias propriedades dos controles e executar as ações apropriadas. A seguinte declaração executa uma macro nomeada PrintReport se a caixa de verificação (nomeada como CheckBox1) estiver marcada: If

```
CheckBox1.Value = True Then Call PrintReport
```

ExcelVBABokks.indb 259

15/02/2013 14:00:10

260 Parte IV: Como se Comunicar com Seus Usuários Eu discuto este tópico em detalhes no Capítulo 17.

DICA

Eu recomendo que você mude o nome padrão que o VBE dá aos seus controles, para algo mais significativo. Você poderia pensar em nomear a caixa de verificação descrita acima como “cbxPrintReport”. Veja que eu precedi o nome com um prefixo de três letras (para “checkbox” - caixa de verificação), indicando o tipo de controle. É uma questão de gosto se você pensa que fazer isso é uma boa prática.

Um Exemplo de UserForm

O exemplo de UserForm desta seção é uma versão ampliada da macro ChangeCase do início do capítulo. Lembre-se de que a versão original dessa macro muda o texto nas células selecionadas para letras maiúsculas. Esta versão modificada usa um UserForm para perguntar ao usuário qual tipo de alteração fazer: letras maiúsculas, letras minúsculas ou as primeiras em maiúscula.

Essa caixa de diálogo precisa obter algumas informações do usuário: o tipo de mudança a fazer no texto. Porque o usuário tem três escolhas, a sua melhor aposta é a caixa de diálogo, com três controles Botão de Opção. A caixa de diálogo também precisa de mais dois botões: um

botão OK e um botão Cancelar. Clicar no botão OK roda o código que executa o trabalho. Clicar no botão Cancel leva a macro a terminar, sem fazer qualquer coisa.

Esta pasta de trabalho está disponível no site do livro. No entanto, você consegue mais desse exercício se seguir as etapas fornecidas aqui e criá-lo, você mesmo.

Criando o UserForm

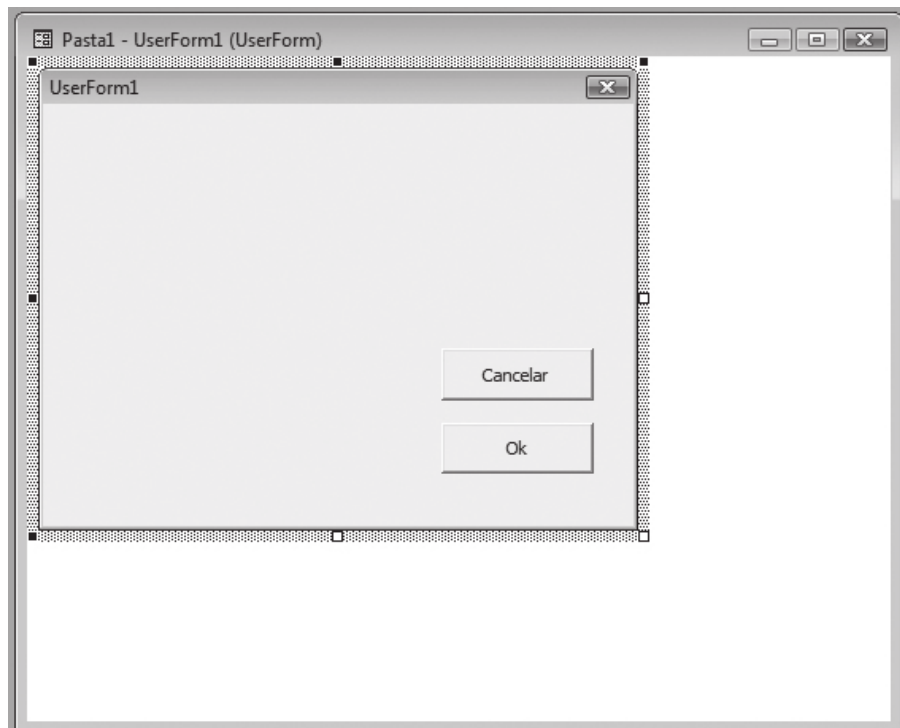
Estas etapas criam o UserForm. Comece com uma pasta de trabalho vazia.

- 1. Pressione Alt + F11 para ativar o VBE.**
 - 2. Se múltiplos projetos estiverem na janela de projeto, selecione aquele que corresponde à pasta de trabalho que você estiver usando.**
 - 3. Escolha (Inserir⇨UserForm).**
- O VBE insere um novo objeto UserForm com uma caixa de diálogo vazia.
- 4. Pressione F4 para exibir a janela Propriedades.**
 - 5. Na janela Propriedades, mude a propriedade Caption para Change Case.**
 - 6. A caixa de diálogo é um pouco grande, assim você pode usar as alças de dimensionamento à direita e embaixo para diminuí-la.**

A etapa 6 também pode ser feita depois que você posicionar todos os controles na caixa de diálogo.

ExcelVBABokks.indb 260

15/02/2013 14:00:10



Capítulo 16: Princípios Básicos de UserForm 261

Adicionando os Botões de comando

Pronto para acrescentar dois Botões de comando — OK e Cancelar — à caixa de diálogo? Me acompanhe:

1. **Assegure-se de que a Caixa de Ferramentas esteja exibida. Se não estiver, escolha Exibir⇨Caixa de Ferramentas.**
2. **Se a janela Propriedades não estiver visível, pressione F4 para exibi-la.**
3. **Na Caixa de Ferramentas, arraste um Botão de comando para a caixa de diálogo, para criar um botão.**

Como você pode ver na caixa Propriedades, o botão tem um nome padrão e legenda: CommandButton1.

4. **Assegure-se de que o Botão de comando esteja selecionado.**

Depois, ative a janela Propriedades e mude as seguintes propriedades:

Propriedade

Mudar para

Name

OKButton

Caption

OK

Default

True

5. Acrescente um segundo objeto Botão de Comando ao UserForm e mude as seguintes propriedades:

Propriedade

Mudar para

Name

CancelButton

Caption *Cancelar*

Cancel *True*

6. Ajuste o tamanho e a posição dos controles, para que a sua caixa de diálogo se pareça um pouco com a Figura 16-4.

Figura 16-4:

O UserForm

com dois

controles

Botão de

comando.

15/02/2013 14:00:10

262 Parte IV: Como se Comunicar com Seus Usuários Adicionando os Botões de opção

Nesta seção, você acrescenta três Botões de opção à caixa de diálogo.

Antes de acrescentar os Botões de opção, adicione um objeto Frame que contém os botões de opção. Um Frame não é necessário, mas faz com que a caixa de diálogo pareça mais profissional.

1. Na Caixa de Ferramentas, clique na ferramenta Quadro e arraste-o para a caixa de diálogo.

Esta etapa cria uma moldura para conter os botões de opção.

2. Use a janela Propriedades para mudar a legenda do quadro para Options.

3. Na Caixa de Ferramentas, clique na ferramenta Botão de opção e arraste-a para a caixa de diálogo (dentro da moldura).

Fazer isso cria um controle Botão de opção.

4. Selecione o Botão de opção e use a janela Propriedades para mudar as seguintes propriedades:

Propriedade

Mudar para

Name

Option/User

Caption

Letras Maiúsculas

Accelerator

U

Value

True

Configurar a propriedade Value para True torna esse Botão de Opção o padrão.

5. Adicione outro Botão de opção e use a janela Propriedades para alterar as seguintes propriedades:

Propriedade

Mudar para

Name

OptionLower

Caption

Letras Minúsculas

Accelerator

L

6. Acrescente um terceiro Botão de opção e use a janela Propriedades para mudar as seguintes propriedades:

Propriedade

Mudar para

Name

OptionProper

Caption

Iniciais Maiúsculas

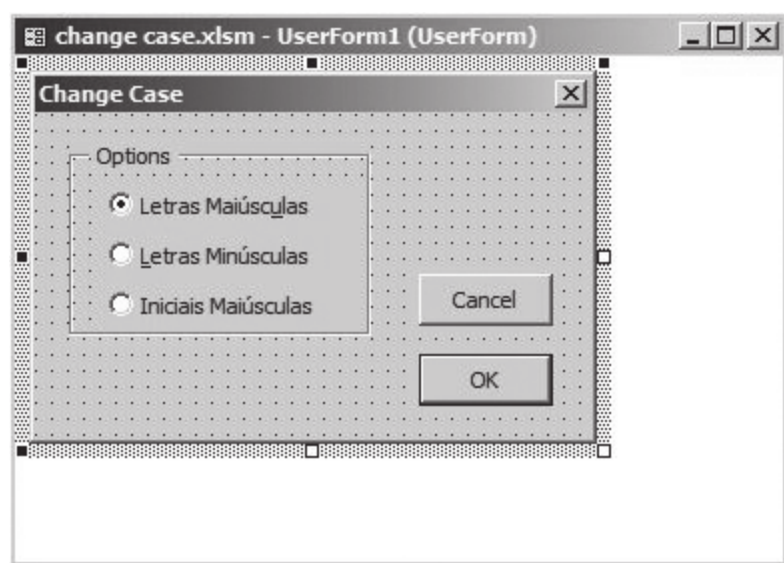
Accelerator

P

7. Ajuste o tamanho e a posição dos botões, do quadra da caixa de diálogo.

O seu UserForm deve se parecer mais ou menos como a Figura 16-5.

15/02/2013 14:00:10



Capítulo 16: Princípios Básicos de UserForm 263

Se você quisesse dar uma espiada para ver como se parece o UserForm quando ele estiver exibido, pressione F5. Nenhum dos controles está funcionando ainda, portanto, você precisa clicar o “X” vermelho na barra de título para fechar a caixa de diálogo.

Figura 16-5:

Este é o

Userform

depois do

acréscimo

dos três

Botões de

opções dentro de um

quadro.

A propriedade Accelerator determina qual letra na legenda é sublinhada— mais importante, ela determina qual combinação Alt-tecla seleciona aquele controle. Por exemplo, você pode selecionar a opção Lower Case (Letra Minúscula) pressionando Alt + L, pois o L está sublinhado.

As teclas de Accelerator são opcionais, porém alguns usuários preferem usar o teclado para navegar pelas caixas de diálogo.

Você pode estar imaginando porque os botões de opção têm teclas de aceleração, mas não os Botões de Comando. Normalmente, os botões OK e cancelar nunca têm teclas de aceleração, porque eles podem ser acessados a partir do teclado. Pressionar Enter é equivalente a clicar OK, pois a propriedade Default do controle é True. Pressionar Esc é equivalente a clicar em Cancelar, pois a propriedade em Cancel do controle é True.

Adicionando procedimentos que lidam com eventos

Chegou a hora do UserForm, de fato, fazer alguma coisa. Eis como acrescentar um procedimento que lida com eventos aos botões Cancelar e OK: **1. Clique duas vezes no botão Cancelar.**

O VBE ativa a janela de código do UserForm e insere um procedimento vazio:

```
Private Sub CancelButton_Click()
```

O procedimento chamado CancelButton_Click é executado quando o botão é clicado, mas só quando a caixa de diálogo é exibida. Em outras palavras, clicar o botão Cancelar quando você estiver

projetando a caixa de diálogo não executa o procedimento. Pelo ExcelVBABokks.indb 263

15/02/2013 14:00:10

264 Parte IV: Como se Comunicar com Seus Usuários fato de a propriedade Cancel do botão Cancelar estar configurada para True, pressionar Esc também dispara o procedimento CancelButton_Click.

2. Insira a seguinte declaração dentro do procedimento (antes da declaração End Sub):

Unload UserForm1

Esta declaração fecha o UserForm (e o remove da memória) quando o botão Cancelar é clicado.

3. Pressione Shift + F7 para voltar ao UserForm.

4. Clique duas vezes no botão OK.

O VBE ativa a janela de código para o UserForm e insere um procedimento Sub chamado

Private Sub OKButton_Click

Quando o UserForm é exibido, clicar OK executa esse procedimento. Porque esse botão tem a sua propriedade Default (Padrão)

configurada para True, pressionar Enter também executa o procedimento OKButton_Click.

5. Entre com o seguinte código dentro do procedimento:

Private Sub OKButton_Click()

Dim WorkRange As Range

‘ Processa apenas células com texto, sem fórmulas

On Error Resume Next

Set WorkRange = Selection.SpecialCells _

(xlCellTypeConstants, xlCellTypeConstants)

‘ Letras Maiúsculas

If OptionUpper Then

For Each cell In WorkRange

cell.Value = Ucase(cell.Value)

Next cell

End If

‘ Letras Minúsculas

If OptionLower Then

For Each cell In WorkRange

cell.Value

=

Lcase(cell.Value)

Next

cell

End If

‘ Iniciais Maiúsculas

If OptionProper Then

For Each cell In WorkRange

cell.Value = Application. _

WorksheetFunction.Proper(cell.Value)

ExcelVBABokks.indb 264

15/02/2013 14:00:10

Capítulo 16: Princípios Básicos de UserForm 265

Next cell

End If

Unload UserForm1

End Sub

O código anterior é uma versão ampliada da macro ChangeCase original que apresentei no início do capítulo. A macro consiste de três blocos de código separados. Este código usa três estruturas If-Then; cada uma tem um loop For Each. Apenas um bloco é executado, determinado pelo botão de opção que o usuário selecionar. A última declaração (*unloads*) fecha a caixa de diálogo depois do trabalho concluído.

Aqui, há alguma coisa meio estranha. Observe que o VBA tem uma função UCase e uma função LCase, mas não tem uma função para

converter texto para as primeiras letras em maiúscula. Portanto, eu uso a função de planilha PROPER do Excel (precedida pela Application.

WorksheetFunction) para a conversão às primeiras letras em maiúscula. Uma outra opção é usar a função VBA StrConv (para detalhes, veja o sistema de Ajuda). A função StrConv não está

disponível em todas as versões do Excel, assim, eu uso a função de planilha PROPER.

Criando uma macro para exibir a caixa de diálogo

Estamos quase terminando com este projeto. A única coisa faltando é uma maneira de exibir a caixa de diálogo. Siga estas etapas para executar o procedimento que faz a caixa de diálogo aparecer: **1. Na janela do VBE, escolha Inserir⇒Módulo.**

O VBE adiciona um módulo VBA vazio (chamado Módulo1) ao projeto.

2. Entre com o seguinte código:

```
Sub ChanceCase()
```

```
If TypeName(Selection) = "Range" Then
```

```
UserForm1.Show
```

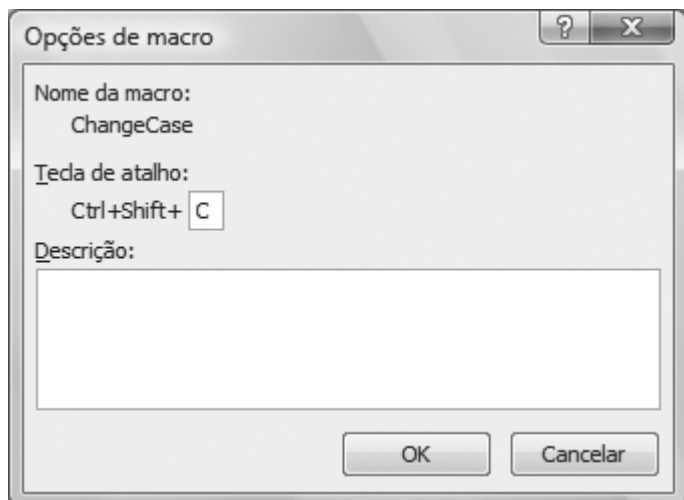
```
Else
```

```
MsgBox "Selecione uma faixa.", vbCritical
```

```
End If
```

```
End Sub
```

Este procedimento é simples. Ele permite se certificar de que uma faixa foi selecionada. Se estiver, a caixa de diálogo é exibida (usando o método Show). Depois, o usuário interage com a caixa de diálogo e o código armazenado no UserForm é executado. Se uma faixa não foi selecionada, o usuário vê uma MsgBox com o texto Selecione uma faixa.



266 Parte IV: Como se Comunicar com Seus Usuários

Como disponibilizar a macro

Nesse ponto, tudo deveria estar funcionando adequadamente. Mas você ainda precisa de uma maneira fácil para executar a macro. Atribua uma tecla de atalho (Ctrl + Shift + C) que executa a macro ChangeCase: 1. Ative a janela do Excel através de Alt + F11.

2. Escolha Desenvolvedor⇒Código⇒Macros ou pressione Alt + F8.

3. Na caixa de diálogo Macros, selecione a macro ChangeCase.

4. Clique no botão Opções.

O Excel exibe a caixa de diálogo Opções de macro.

5. Entre com a letra maiúscula C como tecla de atalho.

Veja a Figura 16-6.

6. Entre com uma descrição da macro no campo Descrição.

7. Clique OK.

8. Clique em Cancelar quando voltar para a caixa de diálogo Macro.

Figura 16-6:

Atribua uma

tecla de
atalho para
executar a
macro

ChangeCase.

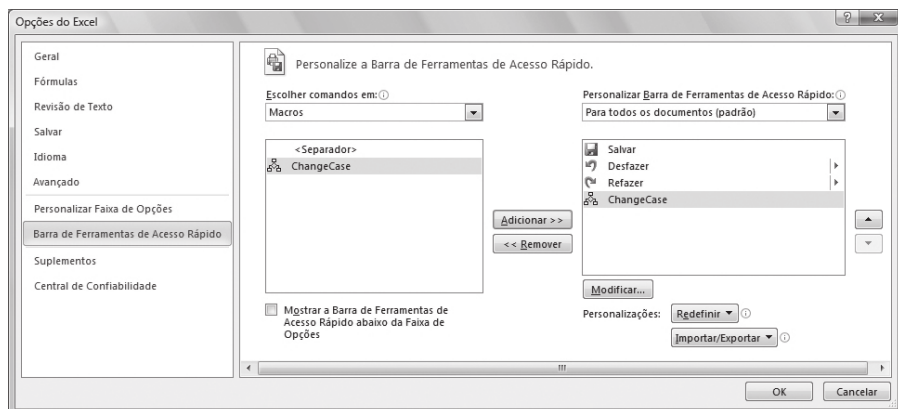
Depois de efetuar esta operação, pressionar Ctrl + Shift + C executa a macro ChangeCase, a qual exibe o UserForm se uma faixa estiver selecionada.

Também é possível disponibilizar essa macro a partir da barra de Ferramentas de Acesso Rápido. Clique com o botão direito na barra de Ferramentas de Acesso Rápido e escolha Personalizar a Barra de Ferramentas de Acesso Rápido. A caixa de diálogo Opções do Excel aparece, e você encontrará a macro ChangeCase relacionada em Macros (veja a Figura 16-7). Adicionar uma macro à sua barra de ferramentas Quick Access funciona muito melhor no Excel 2010. Se a pasta de trabalho que contém a macro não estiver aberta, o Excel 2010

a abre e roda a macro. No Excel 2007, você recebe um erro se a pasta de trabalho não estiver aberta.

ExcelVBABokks.indb 266

15/02/2013 14:00:11





Capítulo 16: Princípios Básicos de UserForm 267

Figura 16-7:

Adicionando a macro

ChangeCase à barra

de ferramentas de

acesso

rápido.

Testando a macro

Finalmente, você precisa testar a macro e a caixa de diálogo, para ter certeza de que elas estão funcionando adequadamente:

- 1. Ative uma planilha (qualquer planilha em qualquer pasta de trabalho).**
- 2. Selecione algumas células que contenham texto.**
- 3. Pressione Ctrl + Shift + C.**

O UserForm aparece. A Figura 16-8 mostra a sua aparência.

- 4. Faça a sua escolha e clique OK.**

Se você fez tudo certo, a macro faz a mudança especificada ao texto nas células selecionadas.

Figura 16-8:

O UserForm

está em

ação.

ExcelVBABokks.indb 267

15/02/2013 14:00:11

	A	B	C
1			
2		JANEIRO	
3		FEVEREIRO	
4		MARÇO	
5		ABRIL	
6		MAIO	
7		JUNHO	
8		JULHO	
9		AGOSTO	
10		SETEMBRO	
11		OUTUBRO	
12		NOVEMBRO	
13		DEZEMBRO	
14			
15	Fórmula:	segunda-feira, 27 de fevereiro de 2012	
16	Data:	segunda-feira, 13 de janeiro de 2003	
17			
18			
19			

268 Parte IV: Como se Comunicar com Seus Usuários

A Figura 16-9 mostra a planilha depois de converter o texto para maiúsculas. Observe que as fórmulas na célula B15 e a data na célula B16 não foram alteradas. Como você se lembra, a macro só trabalha com células que contenha texto.

Figura 16-9:

O texto foi
convertido
para
maiúsculas.

Desde que a pasta de trabalho esteja aberta, você pode executar a macro a partir de qualquer outra pasta de trabalho. Se fechar a pasta de trabalho que contém a sua macro, Ctrl + Shift + C não tem mais qualquer função.

Se a macro não funcionar adequadamente, clique duas vezes as etapas anteriores para localizar e corrigir o erro. Não se preocupe: a depuração é uma parte normal de desenvolver macros. Como um último recurso, faça o download da pasta de trabalho concluída, a partir do site deste livro e tente descobrir o que saiu errado.

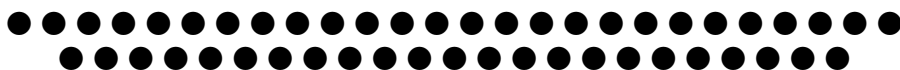
ExcelVBABokks.indb 268

15/02/2013 14:00:11

Capítulo 17

Usando os Controles

de UserForm



Neste Capítulo

- ☐ Como entender cada tipo de controle da caixa de diálogo
- ☐ Mudando as propriedades de cada controle
- ☐ Trabalhando com controles da caixa de diálogo



Um usuário responde a uma caixa de diálogo personalizada (também

conhecida como *UserForm*), usando os vários controles (botões, caixas de edição, botões de edição e assim por diante) que a caixa de diálogo contém. Depois, o seu código VBA utiliza essas respostas para determinar quais ações tomar. Você tem muitos controles à sua disposição e este capítulo fala sobre eles.

Se você trabalhou no exemplo do Capítulo 16, já tem alguma experiência com os controles de UserForm. Este capítulo preenche as lacunas.

Começando com os Controles da Caixa de Diálogo

Nesta seção, falo como acrescentar controles a um UserForm, dar a eles nomes significativos e ajustar algumas de suas propriedades.

Antes de poder fazer qualquer dessas coisas, você deve ter um UserForm, que é obtido escolhendo Inserir ⇨ UserForm no VBE. Quando

você **LEM**

acrescentar um UserForm, assegure-se de que o projeto certo foi selecionado na janela de projeto (se mais de um projeto estiver disponível).

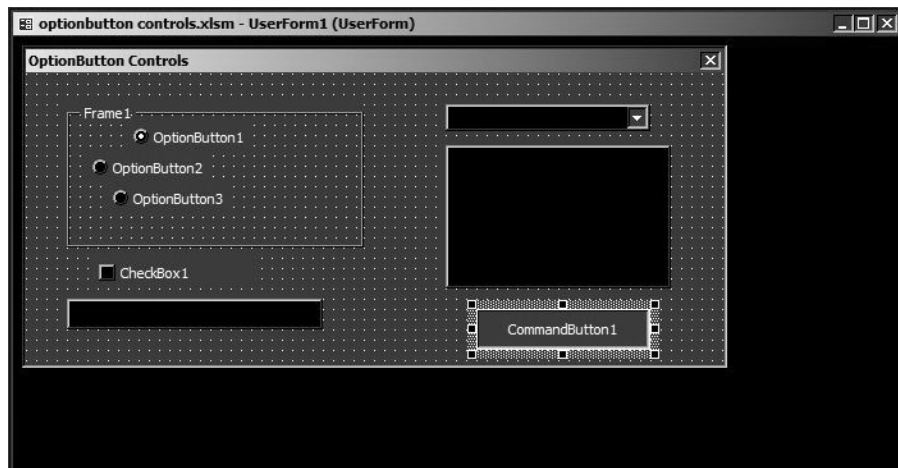
Adicionando controles

Estranhamente, o VBE não tem um menu de comandos que permite

adicionar controles a uma caixa de diálogo. Você deve usar a Caixa de Ferramentas flutuante, que descrevi no Capítulo 16, para acrescentar controles. Normalmente, a Caixa de Ferramentas aparece automaticamente quando você ativa um UserForm no VBE. Se isso não acontecer, você pode exibir a Caixa de Ferramentas escolhendo Exibir⇨Caixa de Ferramentas.

ExcelVBABokks.indb 269

15/02/2013 14:00:11



270 Parte IV: Como se Comunicar com Seus Usuários

Acompanhe, para acrescentar um controle ao UserForm:

1. Clique na ferramenta da Caixa de Ferramentas que corresponde ao controle que deseja adicionar.
2. Clique no UserForm, dimensione e posicione o controle.

Alternativamente, você pode simplesmente arrastar um controle da Caixa de Ferramentas para o UserForm, para criar um controle com as dimensões padrão. A Figura 17-1 mostra um UserForm que contém alguns controles.

Figura 17-1:

Um

UserForm

com alguns

controles

acrescentados.

Um UserForm pode conter grades de linhas verticais e horizontais, as
DICA

quais ajudam a alinhar os controles que você acrescenta. Quando um controle é adicionado ou movido, ele *muda repentinamente* para a grade. Se você não gostar desse recurso, pode desativar as grades,

seguindo estas etapas: **1. Escolha Ferramentas⇒Opções no VBE.**

2. Na caixa de diálogo Opções, selecione a guia Geral.

3. Configure as opções desejadas na seção Configurações da Grade do Formulário.

Introduzindo propriedades de controle

Cada controle que você adiciona a um UserForm tem propriedades que determinam como o controle se parece e se comporta. É possível alterar as propriedades de um controle nas seguintes ocasiões: 5 Por ocasião do projeto – quando você estiver projetando o UserForm. Isso é feito manualmente, usando a janela

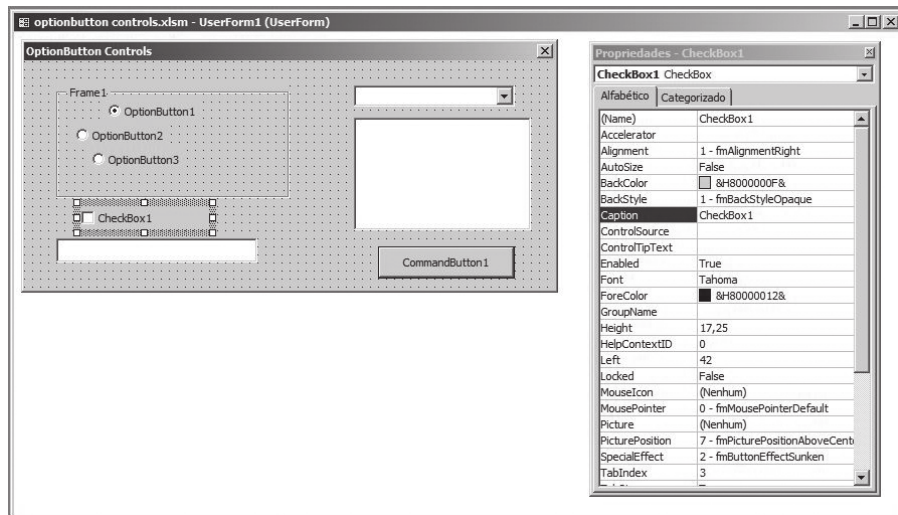
Propriedades.

5 Em tempo de execução – enquanto a sua macro está rodando.

Isso é feito manualmente. As mudanças feitas em tempo de execução são sempre temporárias; elas são feitas na caixa de diálogo que você está exibindo, não no objeto UserForm que você criou.

ExcelVBABokks.indb 270

15/02/2013 14:00:11



Capítulo 17: Usando os Controles de UserForm 271

Quando você acrescenta um controle a um UserForm, quase sempre é necessário fazer alguns ajustes de tempo de execução às propriedades dele. Essas alterações são feitas na janela Propriedades (para exibir a janela Propriedades, pressione F4). A Figura 17-2 mostra a janela Propriedades, a qual exibe propriedades para o objeto selecionado no UserForm – que, por acaso, é um controle CheckBox.

Figura 17-2:

Use a janela

Propriedades para

fazer

alterações

no tempo de

execução às

propriedades

do controle.

Para mudar as propriedades do controle no tempo de execução, você **BRE-SE**

deve escrever código VBA. Por exemplo, você pode querer ocultar um **LEM**

controle em especial quando o usuário clicar uma caixa de verificação.

Nesse caso, você escreve código para alterar a propriedade Visible do controle.

Cada controle tem o seu próprio conjunto de propriedades. No entanto, todos os controles compartilham algumas propriedades comuns, tais como Name (nome), Width (largura) e Height (altura). A Tabela 17-1 relaciona algumas das propriedades comuns disponíveis a muitos controles.

Tabela 17-1

Propriedades Comuns de Controle

Propriedade

O Que Ela É

Accelerator

É a letra sublinhada na legenda do controle. O usuário

pressiona esta tecla junto com a tecla Alt para

selecionar o controle.

AutoSize

Se True, o controle se redimensiona

automaticamente, com base no texto de sua legenda.

BackColor

A cor de fundo do controle.

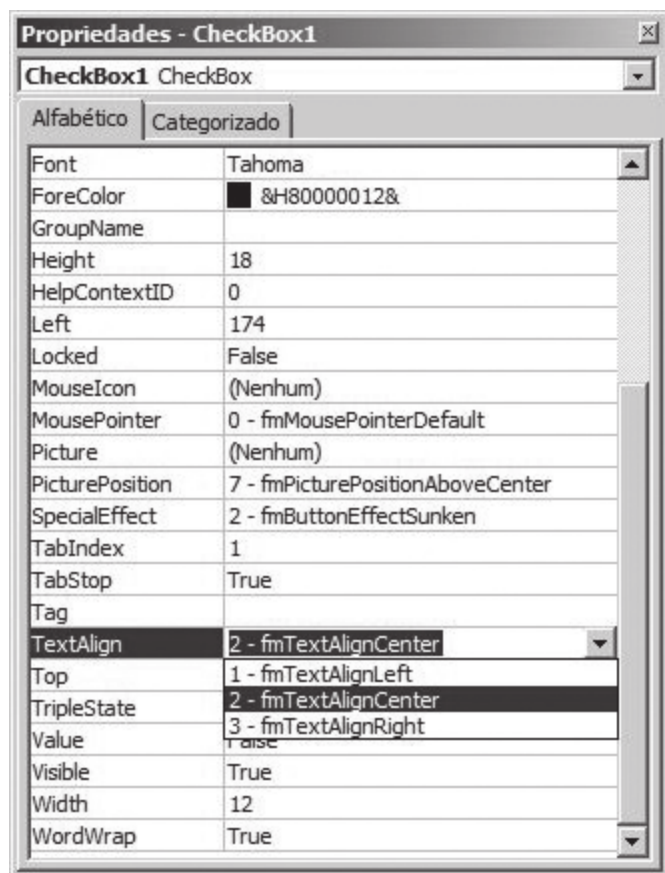
BackStyle

O estilo de fundo (transparente ou opaco).

(continua)

ExcelVBABokks.indb 271

15/02/2013 14:00:11



272 Parte IV: Como se Comunicar com Seus Usuários

Tabela 17-1

(continuação)

Propriedade

O Que Ela É

Caption

O texto que aparece no controle.

Value

O valor do controle.

Left e Top

Valores que determinam a posição do controle

Width e Height

Valores que determinam a largura e altura do

controle.

Visible

Se Falso, o controle fica oculto.

Name

O nome do controle. Por padrão, o nome de um

controle é baseado no tipo de controle. Você pode

mudar o nome para qualquer nome válido, mas

cada nome de controle deve ser único dentro da

caixa de diálogo.

Picture

Uma imagem gráfica para exibir. A imagem pode ser

de um arquivo de gráficos ou você pode selecionar

a propriedade Picture e colar uma imagem que

copiou na Área de Transferência.

Quando você seleciona um controle, tais propriedades do controle aparecem na janela Propriedades. Para alterar uma propriedade, basta selecioná-la na janela Propriedades e fazer a mudança. Algumas propriedades oferecem alguma ajuda. Por exemplo, se você precisar mudar a propriedade TextAlign, a janela Propriedades exibe uma lista drop-down que contém todos os valores válidos de propriedade, conforme mostrado na Figura 17-3.

Figura 17-3:

Mude

algumas

propriedades,

selecionando a partir

de uma lista

drop-down,

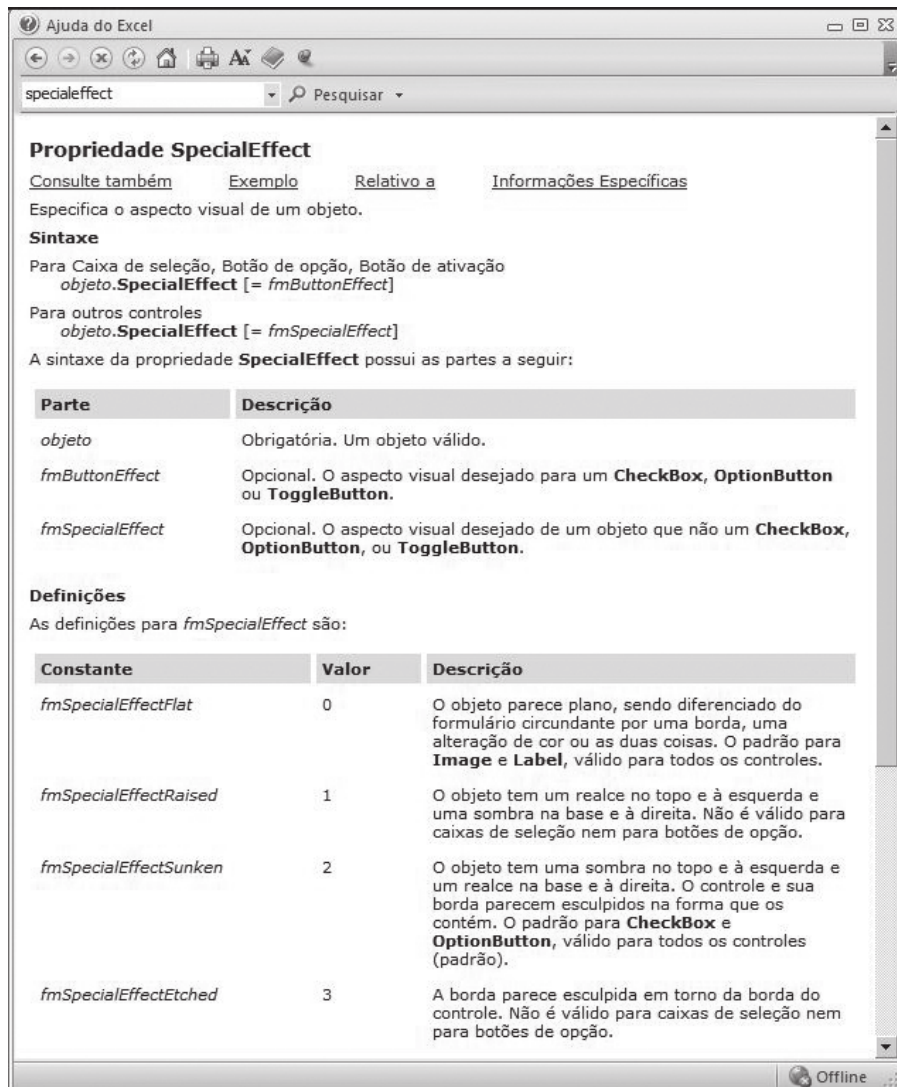
os valores

válidos da

propriedade.

ExcelVBABokks.indb 272

15/02/2013 14:00:11



Capítulo 17: Usando os Controles de UserForm

273

Controles de Caixa de Diálogo:

Os Detalhes

Nas seções a seguir, eu o apresento a cada tipo de controle que você pode usar em caixas de diálogo personalizadas e discuto algumas das propriedades mais úteis. Não abordo cada propriedade para cada

controle, porque isso exigiria um livro quase que quatro vezes maior que este (e seria um livro bem chato).

O sistema de ajuda para os controles e propriedades é minucioso. Para **DICA**

encontrar detalhes completos para uma propriedade em especial, selecione a propriedade na janela Propriedades e pressione F1. A Figura 17-4 mostra a ajuda online para a propriedade SpecialEffect — útil para quando você fizer a sua próxima aventura estrondosa de UserForm.

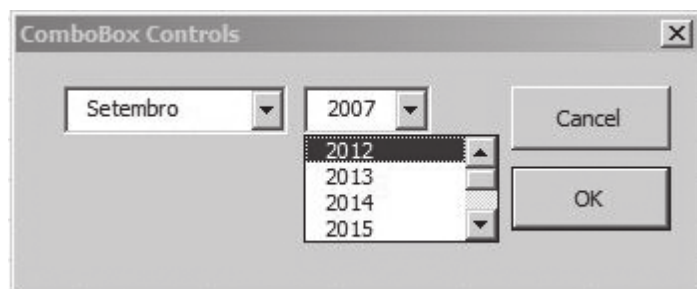
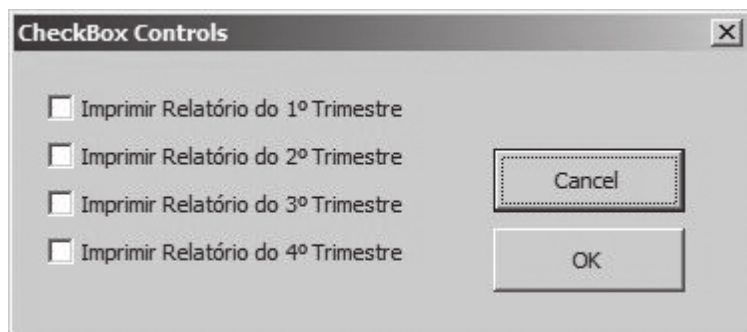
Todos os arquivos de exemplo desta seção estão disponíveis no site deste livro.

Figura 17-4:

O sistema
de Ajuda
oferece
muitas
informa-
ções para
cada
propriedade
e controle.

ExcelVBABokks.indb 273

15/02/2013 14:00:12



274 Parte IV: Como se Comunicar com Seus Usuários

Controle Caixa de Seleção

Um controle Caixa de Seleção (ChekBox) é útil para obter uma escolha binária: sim ou não, verdadeiro ou falso, ligado ou desligado, e assim por diante. A Figura 17-5 mostra alguns exemplos dos controles Caixa de Seleção.

Figura 17-5:

Controles

CheckBox.

A seguir, são descritas as propriedades mais úteis de um controle CheckBox:

5 Accelerator: Uma letra que permite ao usuário mudar o valor do controle usando o teclado. Por exemplo, se o acelerador for A, pressionar Alt + A altera o valor do controle Caixa de Seleção (de marcado para desmarcado ou de desmarcado para marcado).

5 ControlSource: O endereço de uma célula de planilha que está conectada à Caixa de Seleção. A célula exibe TRUE se o controle estiver marcado, ou FALSE (falso) se o controle não estiver marcado.

5 Value: Se True, a Caixa de Seleção tem uma marca de verificação. Se False, ele não tem uma marca de verificação.

CUIDADO!

Não confunda controles Caixa de Seleção com controles Botão de opção.

Eles são meio parecidos, mas são usados com diferentes objetivos.

Controle Caixa de Combinação

Um controle Caixa de Combinação (ComboBox) é semelhante a um controle Caixa de listagem (ListBox), descrito mais adiante, na seção

“Controle Caixa de listagem”. Entretanto, uma Caixa de combinação é uma lista drop-down que exibe apenas um item de cada vez. Uma outra diferença é que o usuário pode ter permissão de inserir um valor que não aparece na lista de itens. A Figura 17-6 mostra dois controles Caixa de Combinação. O controle à direita (para o ano) está sendo usado, assim, ele exibe a sua lista de opções.

Figura 17-6:

Controles

Caixa de

Combinação.

ExcelVBABokks.indb 274

15/02/2013 14:00:12



Capítulo 17: Usando os Controles de UserForm

275

A seguir está uma descrição de algumas propriedades úteis do controle Caixa de combinação:

5 **BoundColumn:** Se a lista contém múltiplas colunas, esta propriedade determina qual coluna contém o valor retornado.

5 **ColumnCount:** O número de colunas na lista.

5 **ControlSource:** Uma célula que armazena o valor selecionado na Caixa de combinação.

5 **ListRows:** A quantidade de itens a exibir quando a lista drop-down é acionada.

5 **ListStyle:** A aparência dos itens da lista.

5 **RowSource:** Uma faixa de endereço que contém a lista dos itens

exibidos na Caixa de combinação.

5 Style: Determina se o controle age como uma lista drop-down ou como uma caixa de combinação. Uma lista drop-down não

permite que o usuário entre com um novo valor.

5 Value: O texto do item selecionado pelo usuário e exibido na Caixa de combinação.

Se a sua lista de itens não está em uma planilha, você pode acrescentar itens a um controle Caixa de Combinação, usando o método AddItem.

DE ESPECIALISTA

O

Mais informações sobre esse método estão no Capítulo 18.

P A P

Controle Botão de comando

O Botão de Comando (Command Button) é apenas um botão comum clicável. Ele não tem utilidade a menos que você forneça um procedimento que lida com eventos para executar quando o botão for clicado. A Figura 17-7 mostra uma caixa de diálogo com nove botões de comando. Dois desses botões apresentam uma imagem em clipart (inserida, copiando o clipart e depois colando-o no campo Picture na janela Propriedades).

Figura 17-7:

Controles

de Botão de

comando.

Quando um botão de comando é clicado, ele executa um

procedimento que lida com eventos com um nome que consiste do nome do botão, um sublinhado e a palavra *Click*. Por exemplo, se um botão de comando for chamado MyButton, clicá-lo executa a macro chamada MyButton_

Click. Essa macro é armazenada na janela de código do UserForm.

ExcelVBABokks.indb 275

15/02/2013 14:00:12



276 Parte IV: Como se Comunicar com Seus Usuários

A seguir, está uma descrição de algumas propriedades úteis do controle Botão de Comando:

5 Cancel: Se Verdadeiro, pressionar Esc executa a macro anexada ao botão (apenas um dos botões do formulário deve ter essa

opção configurada para True).

5 Default: Se Verdadeiro, pressionar Enter executa a macro anexada ao botão (novamente: Apenas um botão deve ter essa

opção configurada para True).

Controle Quadro

Um controle Quadro (Frame) inclui outros controles. Ele é usado com objetivos estéticos ou para agrupar logicamente um conjunto de controles. Um quadro é especialmente útil quando a caixa de diálogo contém mais que um conjunto de controles botões de opção (veja “Controle Botão de Opção”, mais adiante neste capítulo.)

A lista a seguir descreve algumas propriedades úteis do controle Quadro:

5 BorderStyle: A aparência do quadro.

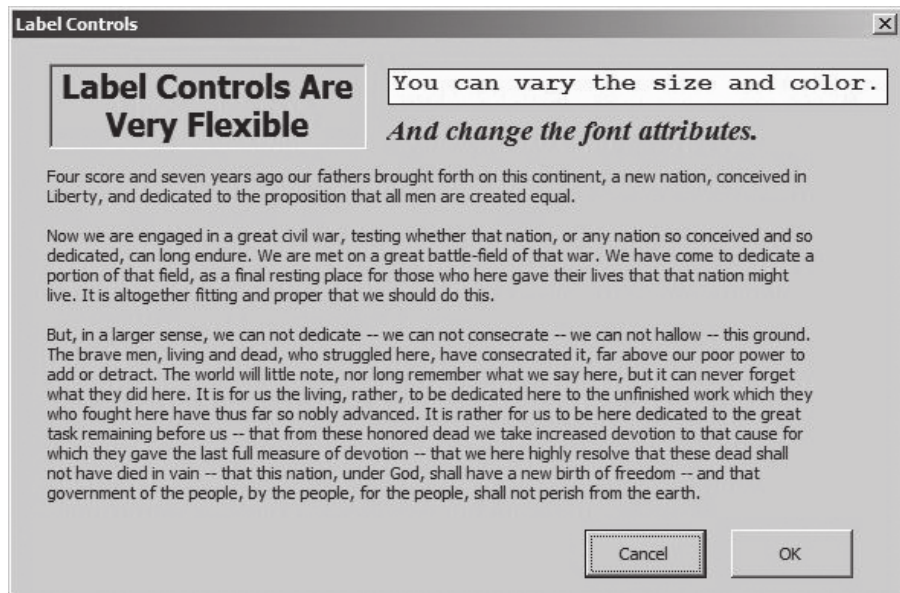
5 Caption: O texto exibido no alto do quadro. A legenda pode ser uma string vazia se você não quiser que o controle exiba uma legenda.

Controle Imagem

Um controle Imagem exibe uma imagem. Você pode usá-lo para exibir a logomarca de sua empresa em uma caixa de diálogo. A Figura 17-8 mostra uma caixa de diálogo com um controle Imagem que exibe a foto de um camarada que escreve livros de Excel.

Figura 17-8:

Um controle
imagem
exibe uma
foto.



Capítulo 17: Usando os Controles de UserForm

277

A seguinte lista descreve as propriedades mais úteis do controle Imagem:

5 Picture: A imagem gráfica que é exibida.

5 PictureSizeMode: Como a imagem é exibida se o tamanho do controle não combinar com o tamanho da imagem.

Ao clicar a propriedade Picture, você é solicitado a fornecer um nome de arquivo. No entanto, a imagem gráfica (quando ela é recuperada) está armazenada na pasta de trabalho. Assim, se você distribui a sua pasta de trabalho para outra pessoa, não tem como incluir uma cópia do arquivo da imagem.

A coleção clipart do Excel é uma ótima fonte de imagens. Use

DICA

(Inserir⇒Ilustrações⇒Clipart) e escolha uma imagem para colocar em sua planilha. Selecione a imagem e pressione Ctrl + C para copiá-la na Área de Transferência. Depois, ative o seu UserForm, clique o controle Imagem e selecione a propriedade Picture na caixa Propriedades.

Pressione Ctrl + V para colar a imagem copiada. Então, você pode apagar a imagem do clipart na planilha.

Algumas imagens gráficas são muito grandes e podem fazer o tamanho **CUIDADO!**

da sua pasta de trabalho aumentar dramaticamente. Para melhores resultados, use uma imagem que seja tão pequena quanto possível.

Controle Rótulo

Um controle Rótulo (Label) simplesmente exibe texto em sua caixa de diálogo. A Figura 17-9 mostra alguns controles Rótulo. Como você pode ver, é possível ter um bom controle na formatação de um controle Rótulo.

Figura 17-9:

Controles

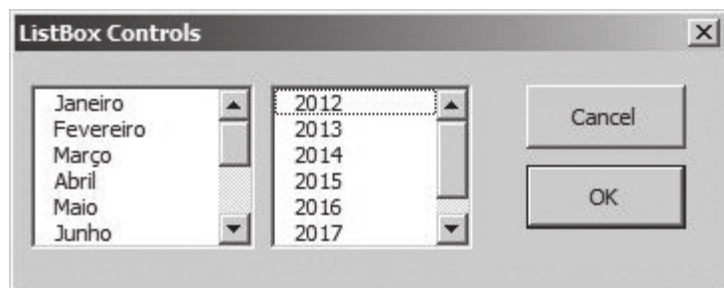
Rótulo são

facilmente

moldados.

ExcelVBABokks.indb 277

15/02/2013 14:00:12



278 Parte IV: Como se Comunicar com Seus Usuários

Controle Caixa de Listagem

O controle Caixa de listagem (ListBox) apresenta uma relação de itens, a partir dos quais o usuário pode escolher um ou mais. A Figura 17-10

mostra uma caixa de diálogo com dois controles Caixa de listagem.

Figura 17-10:

Controles

Caixa de

listagem

Os controles Caixa de listagem são muito flexíveis. Por exemplo, você pode especificar uma faixa de planilha que contém os itens da lista, e a faixa pode consistir de múltiplas colunas. Ou, você pode preencher a lista com itens usando código VBA (eu contei que prefiro esse método?) Se uma Caixa de listagem não exibir todos os itens da lista, aparece uma barra de rolagem para que o usuário possa rolar para baixo, para ver mais itens.

A relação abaixo é uma descrição das propriedades mais úteis do controle:

5 **BoundColumn:** Se a lista contém múltiplas colunas, essa propriedade determina qual coluna contém o valor retornado.

5 **ColumnCount:** A quantidade de colunas na lista.

5 **ControlSource:** Uma célula que o valor selecionado na Caixa de listagem.

5 **IntegralHeight:** É True se a altura da Caixa de listagem se ajustar automaticamente para exibir linhas repletas de texto

quando a lista é rolada verticalmente. Se False, a caixa de

listagem pode exibir linhas parciais de texto quando for rolado verticalmente. Note que, quando essa propriedade é True, a

altura real da lista pode ser ligeiramente diferente, quando o seu UserForm é mostrado, a partir de como você o configurou

originalmente. O Visual Basic pode ajustar a altura para garantir que a última entrada seja totalmente visível.

5 ListStyle: A aparência dos itens da lista.

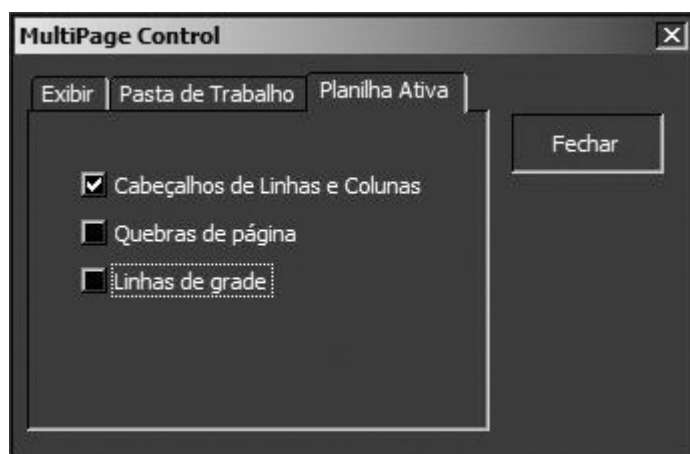
5 MultiSelect: Determina se o usuário pode selecionar múltiplos itens da lista.

5 RowSource: Uma faixa de endereço que contém a lista de itens exibida na Caixa de listagem.

5 Value: O texto do item selecionado na lista.

ExcelVBABokks.indb 278

15/02/2013 14:00:12



Se a Caixa de listagem tiver a sua propriedade MultiSelect configurada **BRE-SE**

para 1 ou 2, então o usuário pode selecionar múltiplos itens na lista.

LEM

Nesse caso, você não pode especificar um ControlSource; é preciso escrever uma macro que determine quais itens são selecionados. O

Capítulo 18 mostra como fazer.

Controle Multipágina

Um controle Multipágina (MultiPage) permite que você crie caixas de diálogo tabuladas, como a caixa de diálogo Formatar Células (que aparece quando você pressiona Ctrl + 1 no Excel). A Figura 17-11 mostra um exemplo de uma caixa de diálogo personalizada que usa um controle Multipágina. Esse controle, em particular, tem três páginas, ou tabulações.

Figura 17-11:

Use o

controle

Multipágina

para criar

uma caixa

de diálogo

tabulada.

A seguir, veja as propriedades mais úteis do controle Multipágina: 5

Style: Determina a aparência do controle. Os tabuladores podem aparecer normalmente (no alto), à esquerda, como botões ou ocultos (sem tabuladores – o seu código VBA

determina qual página é exibida).

5 **Value:** Determina qual página ou tabulador é exibido. Um Valor de

O exibe a primeira página, um Valor de 1 exibe a segunda página e assim por diante.

Por padrão, um controle Multipágina tem duas páginas. Para acrescentar

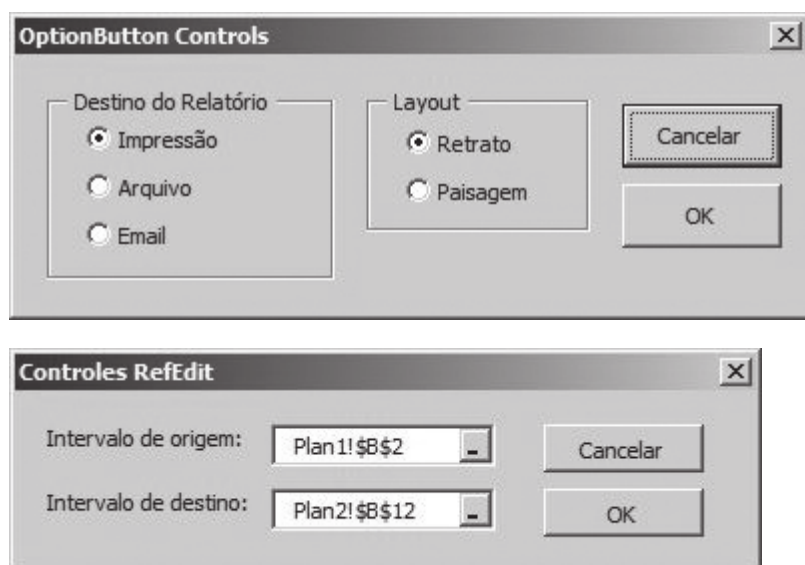
páginas, clique com o botão direito no tabulador e selecione Nova Página no menu Contexto.

Controle Botão de Opção

Os botões de opção (Option Buttons) são úteis quando o usuário precisa selecionar a partir de uma pequena quantidade de itens. Botões de opção são sempre usados em grupos de pelo menos dois. A Figura 17-12 mostra dois conjuntos de botões, e cada conjunto está contido em um quadro.

ExcelVBABokks.indb 279

15/02/2013 14:00:13



280 Parte IV: Como se Comunicar com Seus Usuários

Figura 17-12:

Dois

conjuntos de

controles

botões de

opção, cada

um contido

em um

controle

Frame.

A seguir está uma descrição das propriedades mais úteis do controle Botões de opção:

5 Accelerator: Uma letra que permite ao usuário selecionar a opção usando o teclado. Por exemplo, se o acelerador for um

botão de opção C, então, pressionar Alt + C seleciona o controle.

5 GroupName: Um nome que identifica um botão de opção como sendo associado a outros botões de opção com a mesma

propriedade GroupName.

5 ControlSource): A célula da planilha que está conectada ao botão de opção. A célula exibe TRUE se o controle estiver

selecionado ou FALSE se o controle não estiver selecionado.

5 Value: Se True, o controle está selecionado. Se False, não está selecionado.

Se a sua caixa de diálogo contiver mais que um conjunto de controles, **BRE-SE**

you *deve* mudar a propriedade GroupName em todos os botões de **LEM**

opção em um conjunto em especial. Caso contrário, todos os botões de opção se tornam parte do mesmo conjunto. Alternativamente, você pode encerrar cada conjunto de botões de opção em um controle Quadro, o qual, automaticamente, agrupa os botões de opção em uma moldura.

Controle RefEdit

O controle RefEdit é usado quando você precisa deixar o usuário selecionar uma faixa em uma planilha. A Figura 17-13 mostra um UserForm com dois controles RefEdit. A sua propriedade Value contém o endereço da faixa selecionada (como uma string de texto).

Figura 17-13:

Dois

controles

RefEdit.

ExcelVBABokks.indb 280

15/02/2013 14:00:13



Capítulo 17: Usando os Controles de UserForm

281

Às vezes, o controle RefEdit causa problemas nos UserForms mais complexos. Para melhores resultados, não coloque um controle RefEdit **CUIDADO!**

dentro de um controle Quadro ou Multipágina.

Controle Barra de Rolagem

Quando você adiciona um controle Barra de Rolagem (ScrollBar), pode fazê-lo horizontal ou vertical. A barra de rolagem é semelhante a um botão de rotação (descrito mais adiante). A diferença é que o usuário pode arrastar o botão da barra de rolagem para mudar o valor do controle em aumentos maiores. Uma outra diferença é que, quando você clica o botão up (para cima) em uma barra de rolagem vertical, o valor diminui — o que é um pouco contraintuitivo. Assim, uma barra de rolagem nem sempre é um bom substituto de um botão de rotação.

A Figura 17-14 mostra um controle Barra de rolagem com uma orienta-

ção horizontal. A sua propriedade Value é exibida em um controle Rótulo, colocado abaixo da barra de rolagem.

Figura 17-14:

Uma barra
de rolagem
com um
rótulo abaixo
dela.

A seguir, está uma descrição das propriedades mais úteis de um controle Barra de rolagem:

5 **Value:** O valor atual do controle.

5 **Min:** O valor mínimo do controle.

5 **Max:** O valor máximo do controle.

5 **ControlSource:** A célula de planilha que exibe o valor do controle.

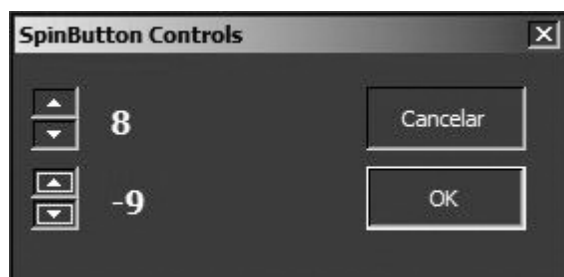
5 SmallChange: A quantia que o valor do controle é alterada por um clique.

5 LargeChange: A quantia que o valor do controle é alterada clicando em qualquer lado do botão.

O controle Barra de rolagem é mais útil para especificar um valor que se expande através de uma ampla faixa de possíveis valores.

ExcelVBABokks.indb 281

15/02/2013 14:00:13



282 Parte IV: Como se Comunicar com Seus Usuários

Controle Botão de Rotação

O controle Botão de Rotação (SpinButton) permite ao usuário selecionar um valor clicando o controle, o qual tem duas setas (uma para aumentar o valor e outra para diminuir o valor). Tal qual uma barra de rolagem, um botão de rotação pode ser orientado horizontal ou verticalmente. A Figura 17-15 mostra uma caixa de diálogo que usa dois botões de rotação verticalmente orientados. Cada controle é conectado ao controle Rótulo à direita (usando procedimentos VBA).

Figura 17-15:

Controles

Botão de

rotação.

As seguintes descrições explicam as propriedades mais úteis de um controle botão de rotação:

5 **Value:** O valor atual do controle.

5 **Min:** O valor mínimo do controle.

5 **Max:** O valor máximo do controle.

5 **ControlSource:** A célula da planilha que exibe o valor do controle.

5 **SmallChange:** A quantia que o valor do controle é alterada por um clique. Normalmente, esta propriedade é configurada para

1, mas você pode fazê-la de qualquer valor.

Se você usar uma ControlSource em um botão de rotação, precisa entender que a planilha é recalculada a cada vez que o valor do contro-CUIDADO!

le é mudado. Portanto, se o usuário mudar o valor de 0 para 12, a planilha é calculada 12 vezes. Se a sua planilha demorar muito para calcular, você pode querer evitar usar uma ControlSource para armazenar o valor.

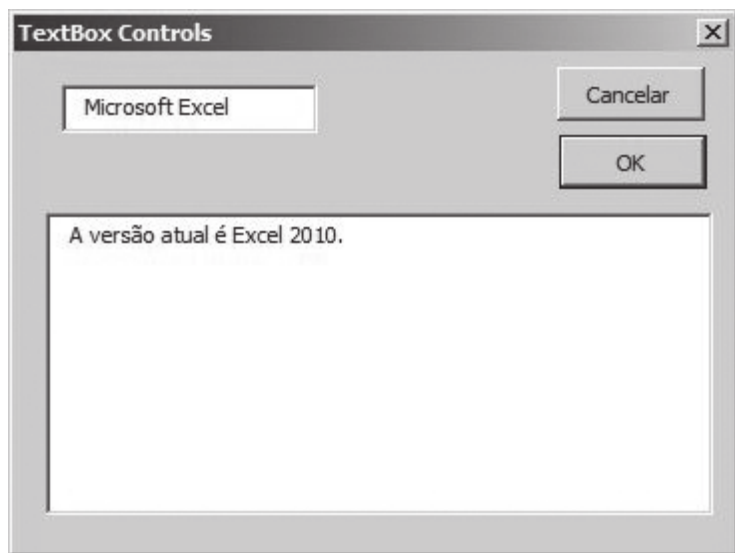
Controle TabStrip

Um controle TabStrip é semelhante a um controle Multipágina, mas ele não é fácil de usar. Na verdade, eu não tenho certeza do motivo pelo qual este controle foi incluído. Você pode muito bem ignorá-lo e, ao invés, usar o controle Multipágina.

Controle Caixa de Texto

Um controle TextBox permite ao usuário entrar com texto. A Figura 17-16

mostra uma caixa de diálogo com duas caixas de texto.



Capítulo 17: Usando os Controles de UserForm

283

Figura 17-16:

Controles

Caixa de

texto.

A seguir está uma descrição das propriedades mais úteis desse controle TextBox:

5 AutoSize: Se True (verdadeiro), o controle ajusta automaticamente o seu tamanho, dependendo da quantidade de texto.

5 ControlSource: O endereço de uma célula que contém o texto no

TextBox.

5 **Integral Height:** Se True, a altura de da caixa de texto se ajusta automaticamente para exibir linhas completas de texto quando a lista é rolada verticalmente. Se False, a caixa de texto pode exibir linhas parciais de texto quando ela é rolada verticalmente.

5 **MaxLenght:** O número máximo de caracteres permitido. Se 0, o número de caracteres é ilimitado.

5 **MultiLine:** Se True, a caixa de texto pode exibir mais de uma linha de texto.

5 **TextAlign:** Determina como o texto é alinhado no TextBox.

5 **WordWrap:** Determina se o controle permite quebra de linha.

5 **ScrollBars:** Determina o tipo de barras de rolagem para o controle: horizontal, vertical, ambos ou nenhum.

Quando você adiciona uma caixa de texto, a sua propriedade WordWrap **DICA**

é configurada para True, e a sua propriedade MultiLine é configurada para False. O resultado? A quebra de linha não funciona! Assim, se você quer quebrar linhas em uma caixa de texto, assegure-se de configurar a propriedade MultiLine para True.

Controle ToggleButton

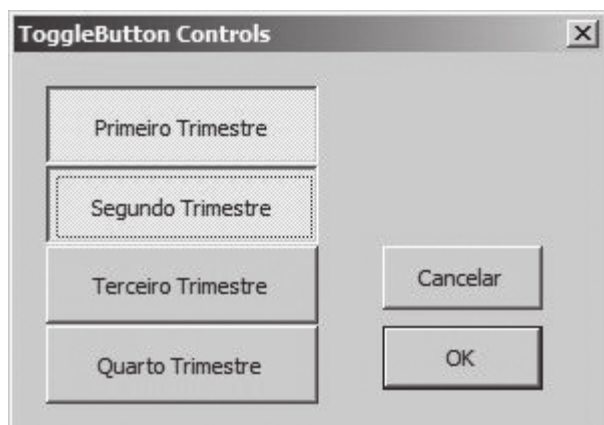
Um controle Botão de Ativação (ToggleButton) tem duas posições: on (ativado) e off (desativado). Clicar o botão alterna entre essas duas posições, e o botão muda a sua aparência quando clicado. O seu valor é True (pressionado) ou False (não pressionado). A Figura 17-17 mostra uma caixa de diálogo com quatro botões de ativação. Os dois

de cima estão alternados.

Raramente eu uso esse controles. Prefiro usar os controles caixa de seleção.

ExcelVBABokks.indb 283

15/02/2013 14:00:13



284 Parte IV: Como se Comunicar com Seus Usuários

Figura 17-17:

Controles

Botão de

ativação.

Trabalhando com Controles

de Caixa de Diálogo

Nesta seção, eu discuto como trabalhar com controles de caixa de diálogo em um objeto UserForm.

Movendo e redimensionando controles

Depois de colocar um controle em uma caixa de diálogo, você o move e redimensiona, usando as técnicas padrão do mouse. Ou, para definir com precisão, você pode usar a janela Propriedades para inserir um valor à propriedade Height, Width, Left ou Top do controle.

É possível selecionar múltiplos controles com Ctrl + clique nos contro-

les. Ou, você pode clicar e arrastar para “laçar” um grupo de controles.

Quando múltiplos controles são selecionados, a janela Propriedades exibe apenas as propriedades comuns a todos os controles selecionados. Você pode mudar essas propriedades comuns, e a alteração será feita em todos os controles que você selecionou, o que é muito mais rápido do que fazer um de cada vez.

Um controle pode ocultar outro; em outras palavras, você pode empilhar um controle sobre outro. A menos que tenha um bom motivo para fazer isso, assegure-se de não sobrepor controles.

Alinhando e espaçando controles

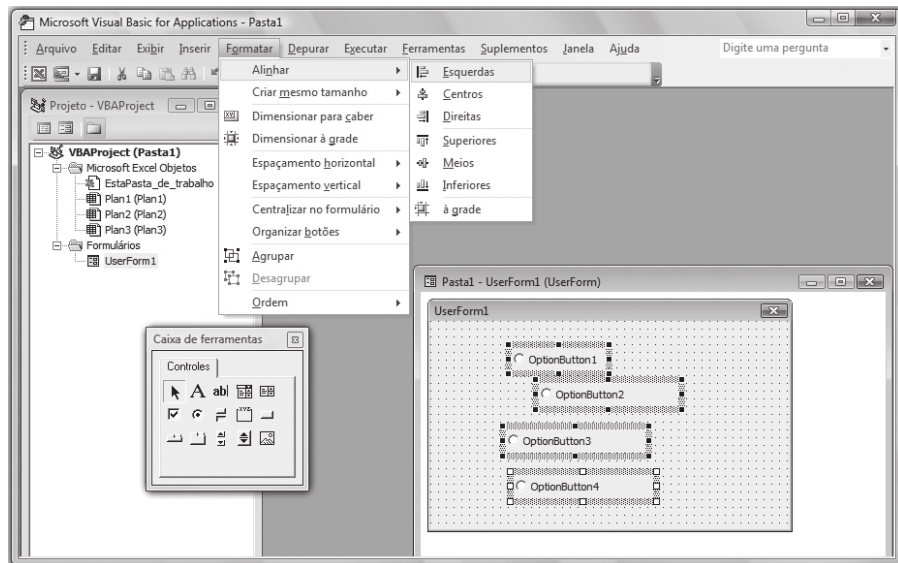
O menu Formatar na janela VBE oferece diversos comandos para

ajudá-lo a alinhar e espaçar com precisão os controles em uma caixa de diálogo. Antes de usar esses comandos, selecione os controles com os quais deseja trabalhar. Esses comandos funcionam exatamente como você poderia esperar, portanto não os explico aqui. A Figura 17-18

mostra uma caixa de diálogo com várias caixas de seleção prontas para ser alinhadas.

ExcelVBABokks.indb 284

15/02/2013 14:00:13



Capítulo 17: Usando os Controles de UserForm

285

Figura 17-18:

Use o comando **Formatar** ➤ **Alinhar** para mudar o alinhamento de controles UserForm.

Quando você seleciona múltiplos controles, o último controle selecionado aparece com alças brancas ao invés das alças pretas normais. O controle com as alças brancas é a base para alinhar ou redimensionar os outros controles selecionados quando você usa o menu **Formatar**.

Acomodando teclado de usuários

Muitos usuários (incluindo os seus, de verdade) preferem navegar através de uma caixa de diálogo, usando o teclado: pressionar **Tab** ou

Shift + Tab permite circular através dos controles, enquanto que pressionar uma tecla de atalho ativa instantaneamente um controle em particular.

Para ter certeza de que as suas caixas de diálogo funcionam adequadamente para os usuários de teclado, você deve estar ciente de duas questões:

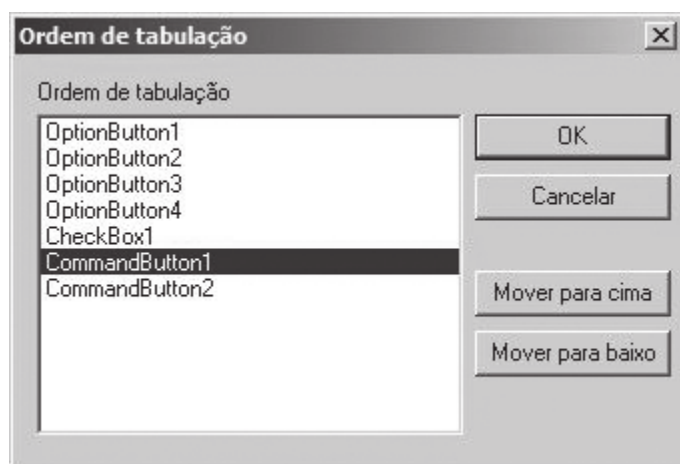
5 A ordem da tabulação

5 Teclas de aceleração

Como mudar a ordem de tabulação

A ordem de tabulação determina a ordem pela qual os controles são ativados quando o usuário pressiona Tab ou Shift + Tab. Ela também determina qual controle tem o *foco* inicial — isto é, qual controle está ativo quando a caixa de diálogo aparece pela primeira vez. Por exemplo, se um usuário estiver inserindo texto em uma caixa de texto, esse ExcelVBABokks.indb 285

15/02/2013 14:00:13



286 Parte IV: Como se Comunicar com Seus Usuários

controle tem o foco. Se o usuário clicar um botão de opção, ele tem o foco. O primeiro controle na ordem de tabulação tem o foco quando o Excel exibe uma caixa de diálogo pela primeira vez.

Para configurar a ordem de tabulação do controle, escolha **Exibir⇨Ordem de tabulação**. Você também pode clicar com o botão direito a caixa de diálogo e escolher **Ordem de tabulação** no menu de atalho. Em qualquer caso, o Excel exibe a caixa de diálogo **Ordem de tabulação** mostrada na Figura 17-19.

Figura 17-19:

A caixa de
diálogo
Ordem de
tabulação.

A caixa de diálogo **Ordem de tabulação** relaciona todos os controles no UserForm. A ordem de tabulação no UserForm corresponde à ordem dos itens na lista. Para mudar a ordem de tabulação de um controle, selecione-o na lista e, depois clique os botões de seta para cima ou para baixo. Você pode escolher mais de um controle (clique enquanto pressiona Shift ou Ctrl) e movê-los todos de uma vez.

Ao invés de usar a caixa de diálogo **Ordem de tabulação**, você pode
DICA

configurar a posição de um controle na ordem de tabulação usando a janela **Propriedades**. O primeiro controle na ordem de tabulação tem uma propriedade **TabIndex** (índice de tabulação) de 0. Se quiser remover um controle da ordem de tabulação, configure a sua propriedade **TabStop** (parar tabulação) para **False**.

Alguns controles (tais como os controles **Quadro** ou **Multipágina**)
agem **BRE-SE**

como contêineres para outros controles. Os controles dentro de um
LEM

contêiner têm suas próprias ordens de tabulação. Para configurar a ordem de tabulação para um grupo de **Botões de opção** dentro de um controle **Quadro**, selecione o quadro antes de escolher o comando **Exibir⇨Ordem de tabulação**.

Configurando teclas de atalho

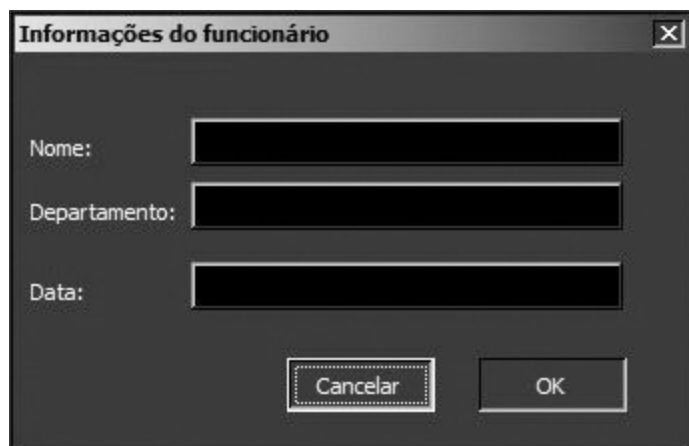
Normalmente, você quer atribuir uma tecla de aceleração, ou *hot key*

(*tecla de atalho*) aos controles da caixa de diálogo. Isso pode ser feito inserindo uma letra para a propriedade Accelerator na janela propriedades. Se um controle não tiver uma propriedade Accelerator (uma caixa de texto, por exemplo), você ainda pode permitir acesso direto de teclado a ela, usando um controle Rótulo. Isto é, designando uma tecla de atalho ao rótulo e o posicionamento antes da caixa de texto na ordem de tabulação.

A Figura 17-20 mostra um UserForm com três Caixas de texto. Os rótulos que as descrevem têm teclas de atalho, e cada rótulo precede a sua caixa de texto correspondente na ordem de tabulação. Pressionar Alt + D, por exemplo, ativa a caixa de texto próxima ao rótulo Department.

ExcelVBABokks.indb 286

15/02/2013 14:00:13



Capítulo 17: Usando os Controles de UserForm

287

Figura 17-20:

Use rótulos

para oferecer

acesso direto

aos controles

que não têm

teclas de

atalho.

Testando um UserForm

O VBE oferece três maneiras de testar um UserForm sem executá-lo a partir de um procedimento VBA:

5 Escolha o comando Executar ÷ Executar Sub/UserForm.

5 Pressione F5.

5 Clique no botão Executar Sub/UserForm na barra de ferramentas padrão.

Quando uma caixa de diálogo é exibida neste modo de teste, você pode experimentar a ordem de tabulação e as teclas de atalho.

Estética de Caixa de Diálogo

Caixas de diálogo podem parecer bonitas, feias ou algo em torno disso.

Uma caixa de diálogo com boa aparência é fácil de se ver, ela tem controles bem dimensionados e alinhados e torna a sua função perfeitamente clara ao usuário. As caixas de diálogo de aspecto feio confundem o usuário, têm controles desalinhados e dão a impressão de que o desenvolvedor não tinha um plano (ou uma ideia).

Tente limitar o número de controles em seu formulário. Se você precisar de muitos controles (uma regra empírica: mais de 10 controles), considere usar um controle Multipágina para separar a tarefa que o usuário tem a fazer, em etapas lógicas (e menores).

Uma boa regra a seguir é tentar fazer suas caixas de diálogo parecidas com as caixas de diálogo integradas do Excel. Na medida em que você adquire mais experiência com a montagem da caixa de diálogo, você pode repetir quase todos os recursos das caixas de diálogo do Excel.

15/02/2013 14:00:14

288 Parte IV: Como se Comunicar com Seus Usuários

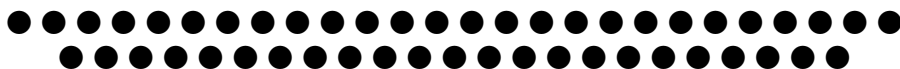
ExcelVBABokks.indb 288

15/02/2013 14:00:14

Capítulo 18

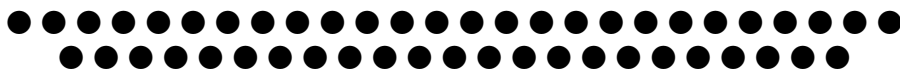
Técnicas e Truques

do UserForm



Neste Capítulo

- ☐ Como usar uma caixa de diálogo personalizada em seu aplicativo
- ☐ Criando uma caixa de diálogo: um exemplo prático



O s capítulo anteriores mostram como inserir um UserForm (o qual contém uma caixa de diálogo personalizada), acrescentar controles ao UserForm e ajustar algumas das propriedades do controle. No entanto, essas habilidades não serão de muita ajuda, a menos que você saiba utilizar Userforms em seu código VBA. Este capítulo oferece esses detalhes que faltam e, no processo, apresenta algumas técnicas e truques úteis.

Como Usar Caixas de Diálogo

Ao usar uma caixa de diálogo em seu aplicativo, normalmente você escreve um código VBA que faz o seguinte:

5 Inicializa os controles UserForm. Por exemplo, você pode escrever código que configura os valores padrão para os controles.

5 Exibe a caixa de diálogo usando o método Show do objeto UserForm.

5 Reage a eventos que ocorrem nos diversos controles.

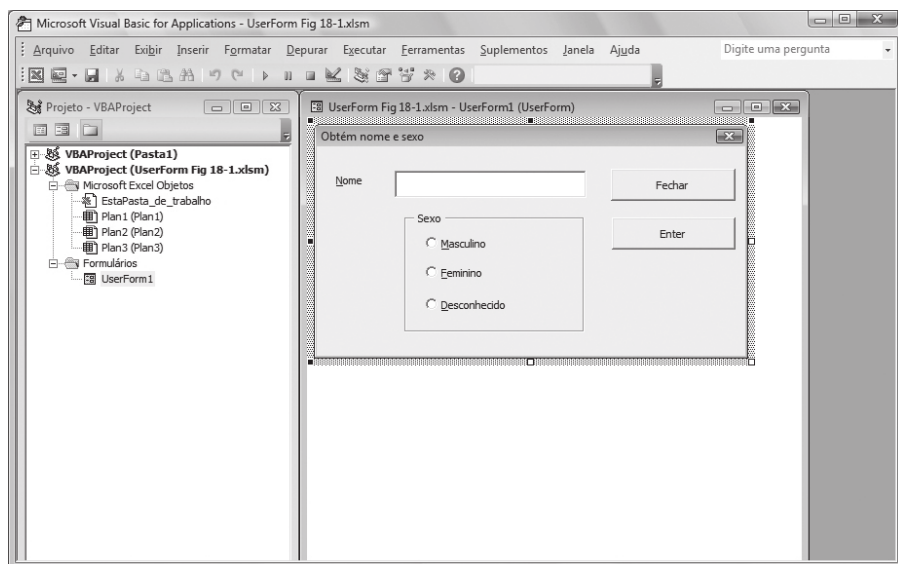
5 Valida as informações fornecidas pelo usuário (se o usuário não cancelou a caixa de diálogo). Esta etapa nem sempre é necessária.

5 Executa alguma ação com as informações fornecidas pelo usuário (se as informações forem válidas).

Um Exemplo de UserForm

O exemplo a seguir demonstra os cinco pontos que descrevi na seção anterior. Nele, você usa uma caixa de diálogo para obter duas partes de informações: o nome e o sexo de uma pessoa. A caixa de diálogo usa um ExcelVBABokks.indb 289

15/02/2013 14:00:14



controle Caixa de Texto para obter o nome e três Botões de Opção, para conseguir o sexo (Masculino, Feminino ou Desconhecido). As informa-

ções coletadas na caixa de diálogo são então enviadas à linha seguinte em branco em uma planilha.

Criando a caixa de diálogo

A Figura 18-1 mostra o UserForm concluído neste exemplo. Para melhores resultados, inicie com uma nova pasta de trabalho contendo apenas uma planilha. Depois, siga estas etapas: **1. Pressione Alt + F11 para ativar o VBE.**

2. Na janela de projeto, selecione uma pasta de trabalho vazia e escolha Inserir⇨UserForm.

Um UserForm vazio é acrescentado ao projeto.

3. Mude a propriedade Caption do UserForm para Get Name and Sex.

Se a janela de propriedade não estiver visível, pressione F4.

Figura 18-1:

Esta caixa de

diálogo pede

ao usuário

para entrar

com um

Nome e

escolher um

Sexo.

Esta caixa de diálogo tem oito controles:

5 Um rótulo. Eu modifiquei as seguintes propriedades para este controle:

Propriedade

Valor

Accelerator

N

Caption

Nome

TabIndex (índice de tabulação)

0

ExcelVBABokks.indb 290

15/02/2013 14:00:14

Capítulo 18: Técnicas e Truques do UserForm

291

5 Uma caixa de texto. Eu modifiquei as seguintes propriedades para este controle:

Propriedade

Valor

Name

TextName (Texto de Nome)

TabIndex

1

5 **Um quadro.** Eu modifiquei as seguintes propriedades para este controle:

Propriedade

Valor

Caption

Sexo

TabIndex

2

5 **Um botão de opção.** Eu modifiquei as seguintes propriedades para este controle:

Propriedade

Valor

Accelerator

M

Caption

Masculino

Name

OptionMale

TabIndex

0

5 **Outro botão de opção:** Eu modifiquei as seguintes propriedades para este controle:

Propriedade

Valor

Accelerator

F

Caption

Feminino

Name

OptionFemale (Opção Feminina)

TabIndex

1

5 **Outro:** Eu modifiquei as seguintes propriedades para este controle:

Propriedade

Valor

Accelerator

D

Caption

Desconhecido

Name

Option Unknown

TabIndex

2

Value (valor)

True

5 Um botão de comando: Eu modifiquei as seguintes propriedades para este controle:

Propriedade

Valor

Caption

Enter

Default

True

Name

EnterButton

TabIndex

3

ExcelVBABokks.indb 291

15/02/2013 14:00:14

**292 Parte IV: Como se Comunicar com Seus Usuários 5 Outro
botão de comando:** Eu modifiquei as seguintes

propriedades para este controle:

Propriedade

Valor

Caption

Fechar

Cancel (cancelar)

True

Name

CloseButton

TabIndex

4

Se você estiver acompanhando em seu computador (e deveria), tome alguns minutos para criar esse UserForm, usando as informações precedentes; assegure-se de criar o objeto Quadro antes de acrescentar a ele os botões de opção.

DICA

Em alguns casos, você pode descobrir que copiar um controle existente é mais fácil que criar um novo. Para copiar um controle, pressione Ctrl enquanto arrasta o controle.

Se você preferir parar de caçar, pode fazer o download do exemplo a partir do site deste livro.

Escrevendo código para exibir a caixa de

diálogo

Depois de ter acrescentado os controles ao UserForm, o seu próximo passo é desenvolver algum código VBA para exibir essa caixa de diálogo: **1. Na janela VBE, escolha Inserir⇒Módulo, para inserir**

um módulo VBA.

2. Entre com a seguinte macro:

```
Sub GetData()
```

```
UserForm1.Show
```

```
End Sub
```

Este curto procedimento usa o método Show do objeto UserForm para exibir a caixa de diálogo.

Disponibilizando a macro

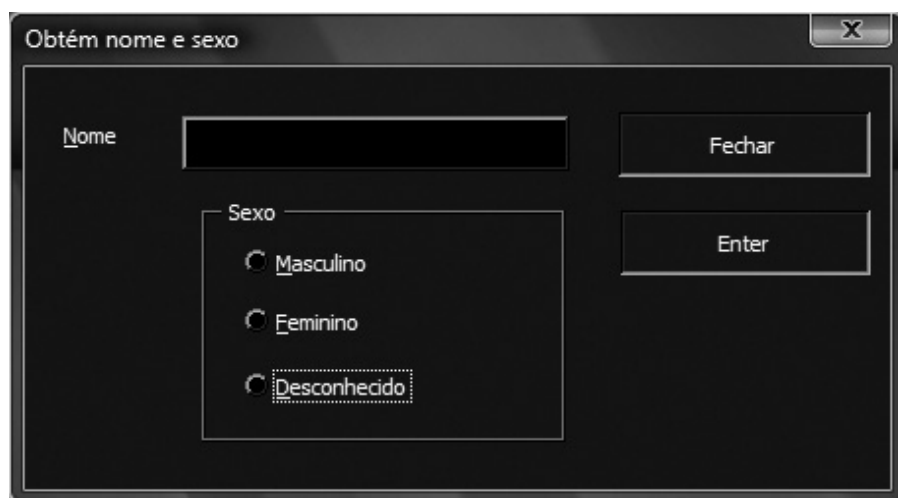
O seguinte conjunto de etapas dá ao usuário uma maneira fácil de executar o procedimento:

1. Ative o Excel.

2. Escolha Desenvolvedor⇒Controles⇒Inserir e clique no ícone do Botão na seção Controle de Formulários.

ExcelVBABokks.indb 292

15/02/2013 14:00:14



The screenshot shows a VBA UserForm titled "Obtém nome e sexo" with a standard Windows-style title bar. The form contains the following elements:

- A text label "Nome" followed by a text input box.
- A text label "Sexo" followed by a group box containing three radio button options: "Masculino", "Feminino", and "Desconhecido". The "Desconhecido" option is currently selected.
- Two buttons on the right side: "Fechar" (Close) and "Enter".

3. Arraste na planilha para criar o botão.

A caixa de diálogo Atribuir macro aparece.

4. Designe a macro GetData ao botão.

5. Edite a legenda do botão para *Data Entry* (entrada de dados).

Se você quiser ser muito extravagante, pode acrescentar um ícone à **DICA**

sua barra de ferramentas de Acesso Rápido. Depois, clicar no ícone roda a macro GetData. Para configurar isso, clique com o botão direito na sua barra de ferramentas de Acesso Rápido e escolha Personalizar Barra de Ferramentas de Acesso Rápido, que exibe guia respectiva na caixa de diálogo Opções do Excel. No menu drop-down, selecione Macros. Modifique o botão e mude o ícone. Se você usa Excel 2010, torne o ícone de acesso rápido visível apenas quando a pasta de trabalho apropriada estiver ativada. Antes de adicionar a macro, use o menu drop-down, do lado superior direito da caixa de diálogo Opções do Excel, para especificar o nome da pasta de trabalho, ao invés de Para todos documentos (Padrão).

Testando a sua caixa de diálogo

Siga estas etapas para testar a sua caixa de diálogo:

1. Clique no botão Data Entry (Entrada de Dados) na planilha. Ou clique no ícone Quick Access da barra de ferramentas, se você configurou um.

A caixa de diálogo aparece, conforme mostrado na Figura 18-2.

Figura 18-2:

Executar o
procedimento GetData
exibe a caixa
de diálogo.

2. Entre com algum texto na caixa de edição.

3. Clique Enter ou Fechar.

Nada acontece — o que é compreensível, pois você ainda não criou quaisquer procedimentos.

4. Clique no botão “X” na barra de título da caixa de diálogo, para se livrar dela.

ExcelVBABokks.indb 293

15/02/2013 14:00:14

294 Parte IV: Como se Comunicar com Seus Usuários *Adicionando procedimentos que lidam com eventos*

Nesta seção, eu explico como escrever os procedimentos que lidam com os eventos que ocorrem quando a caixa de diálogo é exibida.

1. Pressione Alt + F11 para ativar o VBE e, depois, assegure-se de que o UserForm seja exibido.

2. Clique duas vezes no botão Close (fechar) no UserForm.

O VBE ativa a janela Code (código) para o UserForm e oferece um procedimento vazio, chamado CloseButton_Click (Fechar Botão, Clicar).

3. Modifique o procedimento como a seguir:

```
Private Sub CloseButton_Click()
```

```
Unload UserForm1
```

```
End Sub
```

Este procedimento, que é executado quando o usuário clicar no botão Fechar, simplesmente remove a caixa de diálogo da memória.

4. Pressione Shift + F7 para exibir novamente UserForm1.

5. Clique duas vezes no botão Enter e entre com o seguinte procedimento:

Private Sub EnterButton_Click()

Dim Next Row As Long

‘ Certifique-se que Plan1 está ativa

Sheets(“Plan1”).Activate

‘ Determine a próxima linha vazia

NextRow = Application.WorksheetFunction. _

CountA(Range(“A:A”)) + 1

‘ Transfira o nome

Cells(NextRow, 1) = TextName.Text

‘ Transfira o sexo

If OptionMale Then Cells(NextRow, 2) = “Masculino”

If OptionFemale Then Cells(NextRow, 2) = “Feminino”

If OptionUnknown Then Cells(NextRow, 2) = _

“Desconhecido”

‘ Apaga os controles para a próxima entrada

TextName.Text = “ “

OptionUnknown = True

TextName.SetFocus

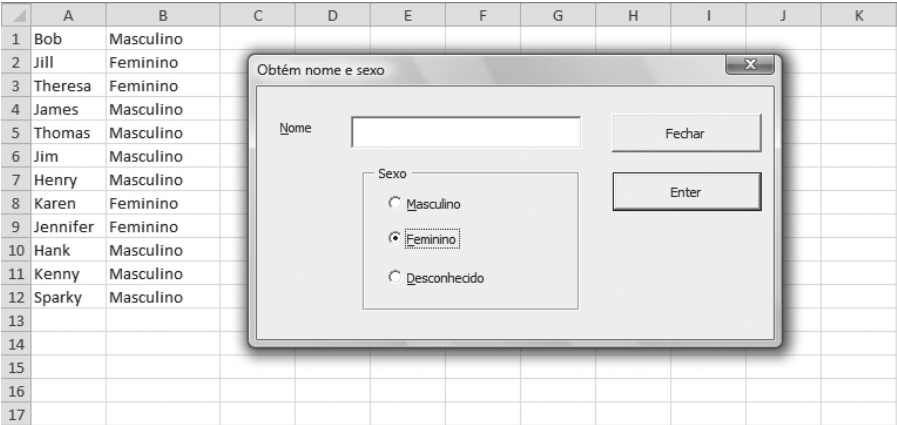
End Sub

6. Agora, ative o Excel e rode novamente o procedimento, clicando o botão Data Entry.

A caixa de diálogo funciona muito bem. A Figura 18-3 mostra como fica em ação.

ExcelVBABokks.indb 294

15/02/2013 14:00:14



Capítulo 18: Técnicas e Truques do UserForm

295

Figura

18-3: Uso

da caixa

de diálogo

personalizada para

entrada de

dados.

Eis como funciona o procedimento `EnterButton_Click`:

5 Primeiro, o código garante que a planilha apropriada (`Plan1`) esteja ativa.

5 Depois, ele usa a função `COUNTA` do Excel para contar o número de entradas na coluna A e determinar a próxima célula em branco na coluna.

5 Em seguida, o procedimento transfere o texto de `TextBox` (Caixa de Texto) para a Coluna A.

5 Depois, ele usa uma série de declarações `If` para determinar qual Botão de Opção foi selecionado e escreve o texto apropriado (Feminino, Masculino ou Desconhecido) na coluna B.

5 Finalmente, a caixa de diálogo é reconfigurada, para deixá-la pronta para a próxima entrada. Observe que clicar no botão

Enter não fecha a caixa de diálogo, pois o usuário, provavelmente quer entrar com mais dados. Para encerrar a entrada de dados, clique no botão Fechar.

Validando os dados

Pratique um pouco mais com esta rotina e você descobrirá que a macro tem um pequeno problema: ela não garante que o usuário entra, de fato, com um nome na caixa de texto. O seguinte código — o qual é inserido no procedimento `EnterButton_Click` antes de transferir

o texto para a planilha — garante que o usuário entre com algum texto na caixa. Se ela estiver vazia, uma mensagem aparece e a rotina é interrompida.

‘ Certifique-se que um nome seja inserido

If TextName.Text = “ “ Then

MsgBox “Você deve inserir um nome.”

Exit Sub

End If

ExcelVBABokks.indb 295

15/02/2013 14:00:15

296 Parte IV: Como se Comunicar com Seus Usuários *Agora a caixa de diálogo funciona*

Depois de fazer estas modificações, você descobre que a caixa de diálogo funciona sem falhas. Na vida real, provavelmente você precisaria reunir mais informações do que apenas nome e sexo. No entanto, os mesmos princípios básicos se aplicam. Você só precisa lidar com mais controles UserForm.

Mais uma coisa a lembrar: se os dados não começarem na linha 1 ou se a área de dados contiver quaisquer linhas em branco, a contagem para a variável NextRow (Próxima Fileira) será errada. A função COUNTA está contando o número de células em A1 e, a suposição é de que os dados comecem na célula A1 e não há células em branco acima do último nome na coluna. Eis uma outra forma de determinar a próxima linha vazia: NextRow = Cells(Rows.Count, 1).End(xlUp).Row + 1

A declaração simula ativar a última célula na coluna A, pressionando End, pressionando Seta para cima e, depois descendo uma linha. Se você fizer isso manualmente, o indicador da célula estará na próxima célula varia na coluna A — mesmo se a área de dados não começar na linha 1 e contiver linhas em branco.

Mais Exemplos do UserForm

Provavelmente, eu poderia encher um livro inteiro com dicas interessantes e úteis para trabalhar com caixas de diálogo personalizadas.

Infelizmente, este livro tem um número limitado de páginas, portanto, eu o completo com mais alguns exemplos.

Um exemplo de Caixa de Listagem

Caixas de listagens são controles úteis, porém, trabalhar com elas pode ser um pouco ardiloso. Antes de exibir uma caixa de diálogo que usa uma caixa de listagem, preencha-a com itens. Depois, quando a caixa de diálogo for fechada, você precisa determinar qual(is) item(ns) o usuário selecionou: **BRE-SE**

Ao lidar com ListBoxes, você precisa saber sobre as seguintes propriedades e métodos:

5 AddItem: Você usa este método para acrescentar um item à caixa de listagem.

5 ListCount: Esta propriedade retorna o número de itens da caixa de listagem.

5 ListIndex: Esta propriedade retorna o índice de número do item selecionado ou conjuntos de itens que são selecionados

(apenas seleções individuais). O primeiro item tem um

ListIndex de 0 (não 1).

ExcelVBABokks.indb 296

15/02/2013 14:00:15

5 **MultiSelect:** Esta propriedade determina se o usuário pode selecionar mais de um item da lista.

5 **RemoveAllItems:** Use este método para remover todos os itens da lista.

5 **Selected:** Esta propriedade retorna um array, indicando itens selecionados (aplicável apenas quando são permitidas

múltiplas seleções).

5 **Value:** Esta propriedade retorna o item selecionado em uma lista.

A maioria dos métodos e propriedades que trabalham com caixas de **DICA**

listagem também trabalha com caixas de combinação. Assim, depois de ter descoberto como lidar com caixas de listagem, você pode transferir esse conhecimento para o seu trabalho com caixas de combinação.

Preenchendo uma Caixa de Listagem

Para melhores resultados, comece com uma pasta de trabalho vazia. O exemplo nesta seção supõe o seguinte:

5 Você adicionou um UserForm.

5 O UserForm contém um controle caixa de listagem chamado `ListBox1`.

5 O UserForm tem um Botão de comando chamado OKButton.

5 O UserForm tem um Botão de comando chamado CancelButton,
o qual tem o seguinte procedimento ao ser clicado:

```
Private Sub Cancelbuton_Click()
```

```
Unload UserForm1
```

```
End Sub
```

O seguinte procedimento é armazenado no procedimento Initialize do UserForm:

1. Selecione o seu UserForm e pressione F7 para ativar a sua janela de código.

O VBE exibe a janela de código para o seu formulário e está pronto para você entrar com o código para o evento Initialize.

2. Usando a lista drop-down de Procedure no alto da janela de código, escolha Initialize.

3. Adicione o código de inicialização ao formulário:

```
Sub Userform_Initialize()
```

```
‘ Preencha a caixa de listagem
```

```
With ListBox1
```

```
.AddItem “Janeiro”
```

```
.AddItem “Fevereiro”
```


.AddItem “Março”

.AddItem “Abril”

.AddItem “Maio”

.AddItem “Junho”

ExcelVBABokks.indb 297

15/02/2013 14:00:15

298 Parte IV: Como se Comunicar com Seus Usuários

.AddItem “Julho”

.AddItem “Agosto”

.AddItem “Setembro”

.AddItem “Outubro”

.AddItem “Novembro”

.AddItem “Dezembro”

End With

‘ Selecione o primeiro item da lista

ListBox1.ListIndex = 0

End Sub

Esta rotina de inicialização roda automaticamente, sempre que o seu UserForm for carregado. Assim, quando você usa o método

Show para o UserForm, o código é executado e a sua caixa de

listagem é preenchida com 12 itens, cada um acrescentado através do método AddItem.

4. Insira um módulo VBA e digite um curto procedimento Sub para exibir a caixa de diálogo:

Sub ShowList()

UserForm1.Show

End Sub

Não é obrigatório usar o procedimento Initialize para preencher as suas listas. Isso pode ser feito em um procedimento VBA normal. Usar esse **DE ESPECIALISTA**

O P

procedimento parece ser algo normal para cuidar de uma etapa tão **A P**

comum (ainda que importante). Se você ocultar o UserForm usando UserForm.Hide e depois exibir novamente o formulário (UserForm1.

Show), o evento Initialize não dispara novamente.

Determinando o item selecionado

O código anterior simplesmente exibe uma caixa de diálogo com uma

caixa de listagem preenchida com nomes de meses. O que está faltando é um procedimento para determinar qual item está selecionado na lista.

Acrescente o seguinte ao procedimento OKButton_Click:

```
Private Sub OKButton_Click()
```

```
Dim Msg As String
```

```
Msg = "Você selecionou o item # "
```

```
Msg = Msg & ListBox1.ListIndex
```

```
Msg = Msg & ListBox1.Value
```

```
MsgBox Msg
```

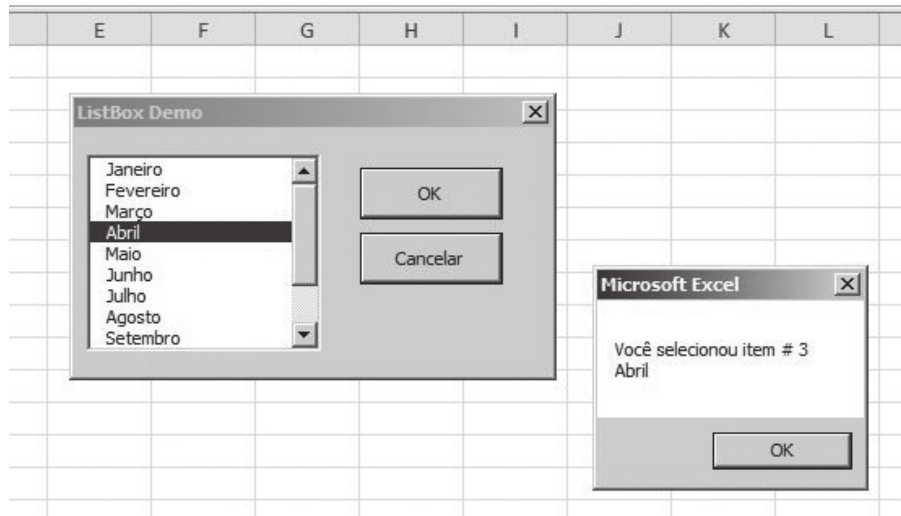
```
Unload UserForm1
```

```
End Sub
```

Este procedimento exibe uma caixa de mensagem com o número do item selecionado e o item selecionado.

ExcelVBABokks.indb 298

15/02/2013 14:00:15



Capítulo 18: Técnicas e Truques do UserForm

299

Se nenhum item da listagem for selecionado, a propriedade `ListIndex` retorna -1. No entanto, esse nunca será o caso com essa caixa de listagem em particular, pois o código no procedimento `UserForm_Initia-lize` selecionou o primeiro item. Assim, *sempre* haverá um item selecionado, se o usuário, de fato, não selecionar um mês.

A Figura 18-4 mostra como isto parece.

Figura 18-4:

Determinando qual item

está

selecionado

em uma

caixa de

listagem.

O primeiro item de uma caixa de listagem tem um `ListIndex` de 0, não 1

(como você poderia esperar). Esse é sempre o caso, ainda que você use **DE ESPECIALISTA**

O P A

uma declaração Option Base 1 (Opção de Base 1) para mudar o limite **P**

inferior padrão para arrays.

Este exemplo está disponível no site deste livro.

Determinando múltiplas seleções

Se a sua caixa de listagem for configurada para que o usuário possa selecionar mais de um item, você descobrirá que a propriedade ListIndex retorna apenas o *último* item selecionado. Para determinar todos os itens selecionados, você precisa usar a propriedade Selected, a qual contém um array.

Para permitir múltiplas seleções em uma caixa de listagem, configure a **BRE-SE**

propriedade MultiSelect para 1 ou 2. Isso pode ser feito usando a janela **LEM**

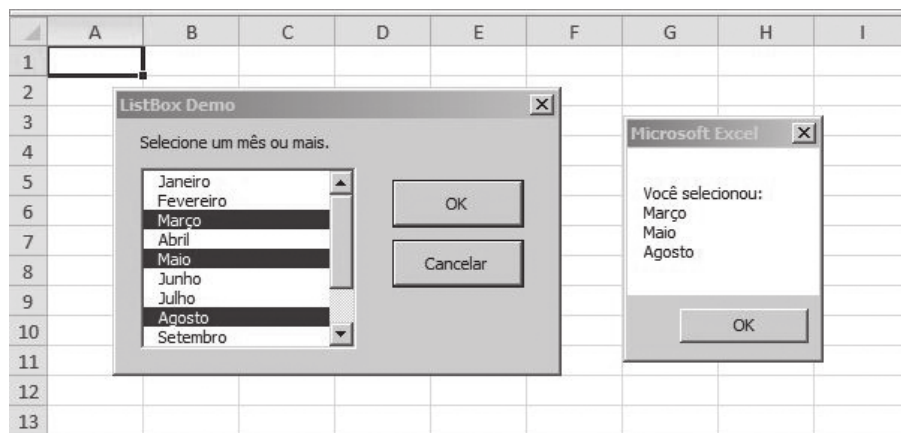
Propriedades ou em tempo de execução, usando uma declaração VBA como esta:

```
UserForm1.ListBox1.MultiSelect = 1
```

A propriedade MultiSelect tem três configurações possíveis. O significado de cada uma é mostrado na Tabela 18-1.

ExcelVBABokks.indb 299

15/02/2013 14:00:15



300 Parte IV: Como se Comunicar com Seus Usuários

Tabela 18-1 Configurações para a propriedade MultiSelect

Valor

Constante VBA

Significado

0

fmMultiSelectSingle Apenas um único item pode ser selecionado.

1

fmMultiSelectMulti Clicar um item ou pressionar a barra de espaço seleciona ou desfaz a seleção de um item na lista.

2

fmMultiSelect

Itens são adicionados ou removidos da seleção

Extended mantendo pressionada a tecla Shift ou

Ctrl enquanto você clica nos itens.

O seguinte procedimento exibe uma caixa de mensagem que relaciona todos os itens selecionados na caixa de listagem. A Figura 18-5 mostra um exemplo.

Figura 18-5:

Determinar

os itens

selecionados em uma

caixa de

listagem

possibilita

múltiplas

seleções.

```
Private Sub OKButton_Click()
```

```
Dim Msg As String
```

```
Dim i As Integer
```

```
Dim Counter As Integer
```

```
Msg = "Você selecionou:" & vbCrLf
```

```
For i = 0 To ListBox1.ListCount - 1
```

```
If ListBox1.Selected(i) Then
```

Counter = Counter + 1

Msg = Msg & ListBox1.List(i) & vbNewLine

End If

Next i

If Counter = 0 then Msg = Msg & “(nada)”

MsgBox Msg

Unload UserForm1

End Sub

ExcelVBABokks.indb 300

15/02/2013 14:00:15

	A	B	C	D	E	F	G	H	I	J	K
1	17	8	10	75	11	64					
2	63	8	22	27	11	54					
3	5	45	70	13	6	48					
4	30	22	15	1	3	15					
5	73	56	87	70	24	87					
6	86	93	69	95	49	92					
7	31	37	30	70	26	93					
8	31	68	1	62	63	87					
9	72	97	58	21	42	61					
10	22	28	92	32	76	58					
11	15	90	55	72	22	1					
12	75	14	21	98	18	9					
13	82	79	43	96	50	15					
14	89	38	92	69	32	33					
15	1	38	93	78	58	45					
16	37	42	11	63	0	60					
17	40	59	14	57	2	37					
18	12	1	14	67	74	46					
19	15	10	25	27	70	36					
20	61	41	84	47	29	71					
21	51	78	21	89	16	95					
22	74	5	31	11	7	58					
23											
24											

RefEdit Demo

Intervalo para negritar:

Sheet1!\$F\$18

Cancelar

OK

Capítulo 18: Técnicas e Truques do UserForm

301

Esta rotina usa um loop For-Next para circular através de cada item na caixa de listagem. Observe que o loop inicia com o item 0 (o primeiro item) e termina com o último item (determinado pelo valor da propriedade ListCount menos 1). Se a propriedade Selected de um item for True, significa que o item da lista foi selecionado. O código também usa uma variável (Counter) para controlar quantos itens são selecionados. Uma declaração If-Then modifica a mensagem se nada estiver selecionado.

Este exemplo está disponível no site deste livro.

Selecionando uma faixa

Em alguns casos, você pode querer que o usuário selecione uma faixa enquanto uma caixa de diálogo é exibida. Um exemplo dessa escolha acontece na caixa de diálogo Criar Tabela, que é exibida quando você escolhe Inserir>Tabela. A caixa de diálogo seleciona a faixa sob a hipótese de que é você vai usar — mas você pode usar esse recurso para mudar a faixa, selecionando células na planilha.

Para permitir a seleção de faixa em sua caixa de diálogo, acrescente

um controle RefEdit. O seguinte exemplo exibe uma caixa de diálogo com a faixa de endereço da região atual exibida em um controle RefEdit, conforme mostrado na Figura 18-6. A região atual é um bloco de células que não estão vazias, que contém a célula ativa. O usuário pode aceitar ou mudar essa faixa. Quando o usuário clicar OK, o procedimento muda a faixa para negrito.

Figura 18-6:

Esta caixa de
diálogo
permite que
o usuário
selecione
uma faixa.

ExcelVBABokks.indb 301

15/02/2013 14:00:15

302 Parte IV: Como se Comunicar com Seus Usuários Este exemplo supõe o seguinte:

5 Você tem um UserForm chamado UserForm1.

5 O UserForm contém um controle botão de comando chamado OKButton.

5 O UserForm contém um controle botão de comando chamado CancelButton.

5 O UserForm contém um controle RefEdit chamado RefEdit1.

O código é armazenado em um módulo VBA e mostrado aqui. Este

código faz duas coisas: inicializa a caixa de diálogo, designando o endereço da região atual ao controle RefEdit e exibe o UserForm.

```
Sub BoldCells()
```

```
‘ Sair se a pasta de trabalho não estiver ativa
```

```
If TypeName(ActiveSheet) <> “Worksheet” Then _
```

```
Exit Sub
```

```
‘ Selecione a região atual
```

```
ActiveCell.CurrentRegion.Select
```

```
‘ Inicialize o controle RefEdit
```

```
UserForm1.RefEdit1.Text = Selection.Address
```

```
‘ Mostrar caixa de diálogo
```

```
UserForm1.Show
```

```
End Sub
```

O seguinte procedimento é executado quando o botão OK é clicado.

Este procedimento executa uma simples verificação de erro para garantir que a faixa especificada no controle RefEdit seja válida.

```
Private Sub OKButton_Click()
```

```
On Error GoTo BadRange
```

```
Range(RefEdit1.Text).Font.Bold = True
```

Unload UserForm1

Exit Sub

BadRange:

MsgBox “A faixa especificada não é válida.”

End Sub

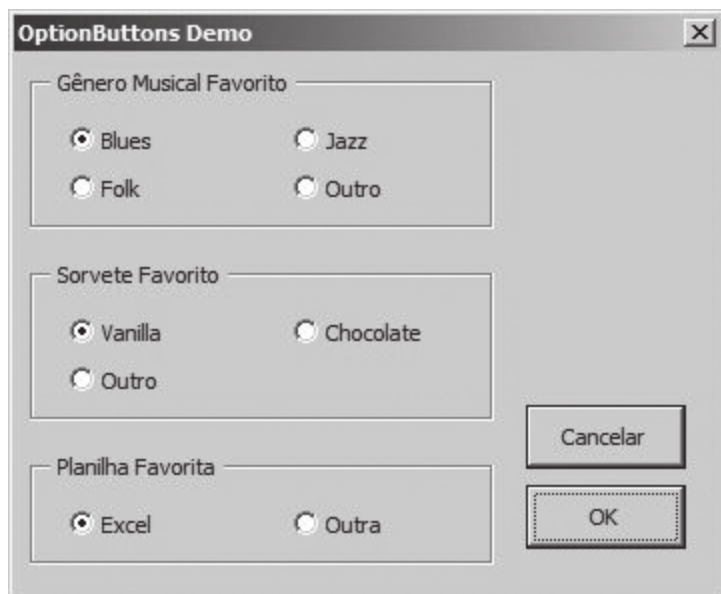
Se ocorrer um erro (mais provavelmente uma especificação inválida de faixa no controle RefEdit), o código pula para a etiqueta BadRange e é exibida uma caixa de mensagem. A caixa de diálogo permanece aberta para que o usuário possa selecionar outra faixa.

Usando múltiplos conjuntos de Botões de opção

A Figura 18-7 mostra uma caixa de diálogo personalizada com três conjuntos de botões de opção. Se o seu UserForm contiver mais do que um conjunto de botões, assegure-se de que cada conjunto trabalhe como um grupo. Você pode fazer isso em uma de duas maneiras:

ExcelVBABokks.indb 302

15/02/2013 14:00:16



Capítulo 18: Técnicas e Truques do UserForm

303

5 Organize cada conjunto de botões em um controle Quadro. Essa abordagem é a mais fácil, e também faz a caixa de diálogo parecer mais organizada. É mais fácil acrescentar o quadro antes de adicionar os botões. No entanto, você também pode arrastá-los para um quadro.

5 Assegure-se de que cada conjunto de botões tenha uma única propriedade `GroupName` (que você especifica na caixa Propriedades). Se os botões estiverem em um quadro, você não precisa se preocupar com a propriedade `GroupName`.

Figura 18-7:

Esta caixa
de diálogo

contém três

conjuntos de

botões de

opção.

Apenas um botão de opção de cada grupo pode ter o valor True. Para **BRE-SE**

especificar uma opção padrão a um conjunto de botões, basta configurar a propriedade Value no item Default para True. Isso pode ser feito diretamente na caixa Propriedades ou usando código VBA:

UserForm1.OptionButton1.Value = True

Este exemplo está disponível no site deste livro. Ele também contém o código que exibe as opções selecionadas quando o usuário clicar OK.

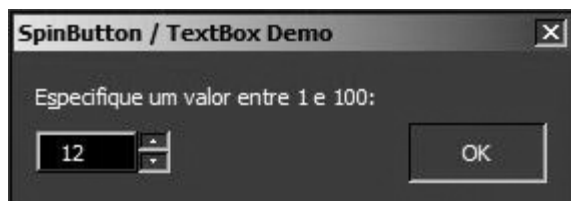
Utilizando um Botão de Rotação e uma

Caixa de Texto

Um controle Botão de rotação permite que o usuário especifique um número, clicando setas. Esse controle consiste apenas em setas (sem texto), portanto você quer um método para exibir o número selecionado. Uma opção é usar um controle Rótulo, mas isso tem uma desvantagem: o usuário não pode digitar texto em um rótulo. Uma escolha melhor é usar uma caixa de texto.

Um controle Botão de rotação e um controle Caixa de texto formam um par natural, e o Excel os usa frequentemente. Por exemplo, verifique a caixa de ExcelVBABokks.indb 303

15/02/2013 14:00:16



diálogo Configurar página do Excel quanto a alguns exemplos. De preferência, Botões de rotação e Caixa de texto estão sempre em sincronia: se o usuário clicar no botão, o seu valor deve aparecer na caixa de texto. E se o usuário entrar com um valor diretamente na caixa de texto, o botão de rotação assumirá aquele valor. A Figura 18-8 mostra uma caixa de diálogo personalizada com um botão de rotação e uma caixa de texto.

Figura 18-8:

Um UserForm

com um

botão de

rotação e

uma

companheira,

a Caixa de

texto.

Este UserForm contém os seguintes controles:

5 Um botão de rotação chamado SpinButton1, com a sua propriedade Min configurada para 1 e sua propriedade Max configurada para 100.

5 Uma caixa de texto chamada TextBox1.

5 Um botão de comando chamado OKButton.

A seguir, o procedimento que lida com eventos para o botão de rotação.

Este procedimento lida com o evento Change, que é disparado sempre

que o valor do botão de rotação é alterado. Quando esse valor mudar (quando ele for clicado), este procedimento designa o seu valor ao TextBox. Para criar este procedimento, clique duas vezes no botão de rotação, para ativar a janela de código do UserForm. Depois, entre com este código: Private Sub SpinButton1_Change()

```
TextBox1.Text = SpinButton1.Value
```

```
End Sub
```

O controlador de eventos da caixa de texto, que é listado a seguir, é um pouco mais complicado. Para criar este procedimento, clique duas vezes na caixa para ativar a janela de código do UserForm. Este procedimento é executado sempre que o usuário mudar o texto na caixa.

```
Private Sub TextBox1_Change()
```

```
Dim NewVal As Integer
```

```
NewVal = Val(TextBox1.Text)
```

```
If NewVal >= Spinbutton1.Min And _
```

```
NewVal <= SpinButton1.Max Then _
```

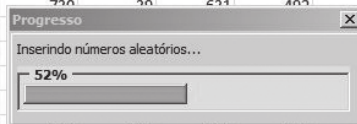
```
SpinButton1.Value = NewVal
```

```
End Sub
```

ExcelVBABokks.indb 304

15/02/2013 14:00:16

	A	B	C	D	E	F	G	H	I	J	K
1	930	81	268	226	480	273	413	878	510	467	226
2	668	114	675	730	30	631	403	435	715	484	535
3	924	971	476					829	238	548	692
4	91	289	319					716	677	681	190
5	83	133	572					857	309	865	650
6	170	511	774					13	419	55	768
7	50	138	342	700	104	215	027	578	881	444	527
8	181	454	522	95	726	721	266	479	982	982	642
9	358	885	591	48	639	915	127	808	504	136	267
10	540	366	312	642	836	639	264	720	49	914	950
11	931	100	637	336	4	640	997	567	618	122	831
12	305	578	658	162	106	842	115	295	441	884	98
13	586	840	814	119	208	861	330	79	53	409	692
14	673	995	340	814	556	773	981	216	622	2	257
15	526	986	972	349	903	126	984	269	530	334	343
16	483	603	903	463	93	208	39	449	197	956	861
17	848	751	978	927	880	561	82	267	162	62	788
18	714	965	158	177	16	749	985	413	410	511	114
19	40	175	213	210	571	370	513	905	951	86	50
20	985	721	682	724	516	325	526	477	646	13	481
21	479	624	73	506	549	333	432	219	289	725	667



Capítulo 18: Técnicas e Truques do UserForm

305

Este procedimento usa uma variável, a qual armazena o texto na caixa de texto (convertido para um valor com a função Val). Depois, ele verifica se o valor está dentro da faixa adequada. Se assim for, o botão de rotação recebe o valor no TextBox. O efeito é que o valor do botão é sempre igual ao valor da caixa (supondo que o valor do botão esteja na faixa apropriada).

Se você usar F8 para uma única etapa através do código no modo de depuração, notará que quando a linha `SpinButton.Value = NewVal` é **DE ESPECIALISTA**

O P

executada, o evento `change` do botão de rotação dispara imediatamente.

A P

Por outro lado, o evento `SpinButton1_Change` configura o valor da caixa de texto1. Por sorte, isso não dispara o evento `TextBox1_Change`, pois o seu valor não é, de fato, alterado pelo evento `SpinButton1_Change`. Mas, você pode imaginar que este efeito pode causar resultados surpreendentes em seu UserForm. Confuso? Lembre-

se apenas que se o seu código mudar o valor de um controle, o evento `Change` daquele controle disparará.

Este exemplo está disponível no site deste livro. Há também alguns sinos e apitos que você pode julgar úteis.

Usando um UserForm como um indicador de progresso

Uma das perguntas mais comuns em programação de Excel que escuto é:

“Como posso exibir o progresso da execução de uma macro comprida?”

Resposta: Use um UserForm para criar um atraente indicador de progresso, conforme mostrado na Figura 18-9. No entanto, o uso de caixas de diálogo exige alguns truques — que estou prestes a mostrar para você.

Figura 18-9:

Este

UserForm

funciona

como um

indicador de

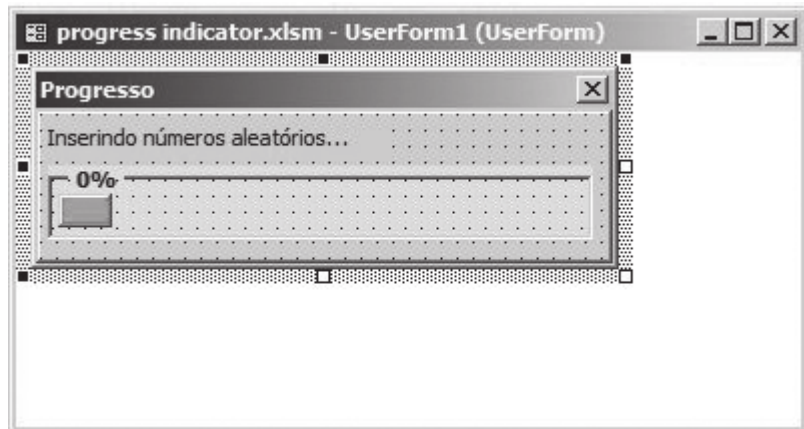
progresso

para uma

macro longa.

ExcelVBABokks.indb 305

15/02/2013 14:00:16



306 Parte IV: Como se Comunicar com Seus Usuários

Criando o indicador de progresso na caixa de diálogo

O primeiro passo é criar o seu UserForm. Neste exemplo, a caixa de diálogo exibe o progresso enquanto uma macro insere números aleatórios em 100 colunas e 1.000 linhas da atraente planilha. Para criar a caixa de diálogo, siga estas etapas:

1. Ative o VBE e insira um novo UserForm.
2. Mude a legenda do UserForm para *Progress*.
3. Acrescente um objeto Quadro e configure as seguintes propriedades:

Propriedade

Valor

Caption

0%

Name

FrameProgress

SpecialEffect

2 – fmSpecialEffectSunken

Width

204

Height

28

4. Adicione um objeto Rótulo dentro do quadro e configure as seguintes propriedades:

Propriedade

Valor

Name

LabelProgress (Progresso de

Etiqueta)

BackColor

&H000000FF& (red)

Caption

(sem legenda)

SpecialEffect

1 – fmSpecialEffectRaised

Width

20

Height

13

Top

5

Left

2

5. Acrescente outro Rótulo no quadro e mude a sua propriedade

Caption para *Entering random numbers...*

O UserForm deve se parecer com a Figura 18-10.

Figura 18-10:

(figura)

O indicador

de progresso

de UserForm.

Os procedimentos

ExcelVBABokks.indb 306

15/02/2013 14:00:16

Capítulo 18: Técnicas e Truques do UserForm

307

Os procedimentos

Este exemplo usa dois procedimentos e um módulo de nível variável.

5 O módulo de nível variável. Localizado em um módulo VBA.

Essa variável contém a cópia do formulário de usuário:

Dim ProgressIndicator as UserForm1

5 EnterRandomNumbers. Ele faz todo o trabalho e é executado quando o UserForm é exibido. Observe que ele chama o

procedimento UpdateProgress, que atualiza o indicador de progresso na caixa de diálogo:

Sub EnterRandomNumbers ()

‘ Inserir números aleatórios na planilha de trabalho ativa

Dim Counter As Long

Dim RowMax As Long, ColMax As Long

Dim r As Long, c As Long

Dim PctDone As Single

‘ Cria uma cópia do formulário em uma variável

Set ProgressIndicator = New UserForm1

‘ Mostra ProgressIndicator em estado sem modo

ProgressIndicator.Show vbModeless

If TypeName(ActiveSheet) <> "Worksheet" Then

Unload ProgressIndicator

Exit Sub

End If

‘ Insira os números aleatórios

Cells.Clear

Counter = 1

RowMax = 200

ColMax = 50

For r = 1 To RowMax

For c = 1 To ColMax

Cells, c) = Int(Rnd * 1000)

Counter = Counter + 1

Next c

PctDone = Counter / (RowMax * ColMax)

Call UpdateProgress(PctDone)

Next r

Unload ProgressIndicator

Set ProgressIndicator = Nothing

End Sub

5 UpdateProgress. Este procedimento aceita um argumento e atualiza o indicador de progresso na caixa de diálogo:

Sub UpdateProgress(pct)

With ProgressIndicator

.FrameProgress.Caption = Format(pct, "0%")

.LabelProgress.Width = pct*(.FrameProgress _

ExcelVBABokks.indb 307

15/02/2013 14:00:16

308 Parte IV: Como se Comunicar com Seus Usuários

.Width – 10)

End With

‘ A declaração DoEvents atualiza o formulário

DoEvents

End Sub

Como este exemplo funciona

Quando o procedimento EnterRandomNumbers é executado, ele carrega uma cópia do UserForm1 no módulo variável chamado ProgressIndicator.

Depois, ele configura a largura do rótulo LabelProgress para 0 e exibe o UserForm na posição Modeless (de modo que o código continuará a rodar).

O procedimento EnterRandomNumber verifica a planilha ativa. Se ela não for uma planilha, o UserForm (ProgressIndicator) é fechado, e o procedimento termina sem ação. Se a planilha ativa é uma planilha, o procedimento faz o seguinte: 1. Apaga todas as células na planilha ativa.

2. Faz loops através das linhas e colunas (especificadas pelas variáveis RowMax e ColMax) e insere um número aleatório.

3. Aumenta a variável Counter e calcula a porcentagem completa (que é armazenada na variável PctDone).

4. Chama o procedimento UpdateProgress, o qual exibe a porcentagem completa, mudando a largura da etiqueta LabelProgress e

atualizando a legenda do controle de quadro.

5. Por fim, o UserForm é fechado.

Claro que usar um indicador de progresso fará a sua macro rodar um pouco mais lentamente, pois o código está fazendo trabalho adicional, atualizando o UserForm. Se a velocidade for absolutamente crítica, pense duas vezes em usar um indicador de progresso.

Se você adaptar esta técnica para o seu próprio uso, precisa descobrir

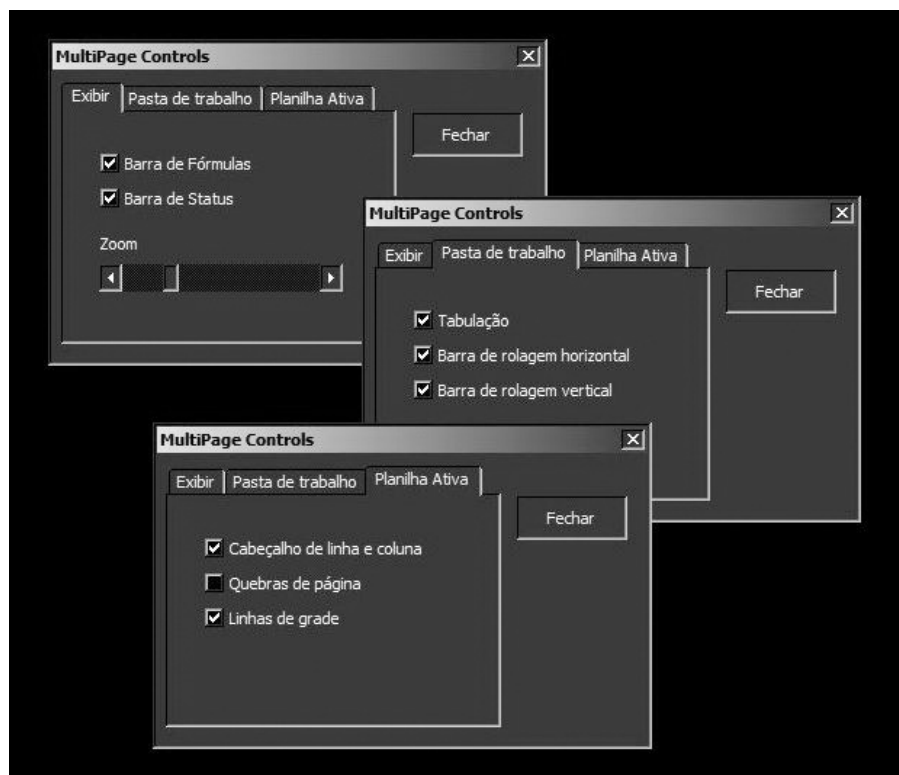
como determinar o progresso da macro, que varia dependendo de sua macro. Depois, chame o procedimento UpdateProgress a intervalos regulares enquanto a sua macro estiver executando.

Este exemplo está disponível no site deste livro.

Criação de uma caixa de diálogo Multipágina

Caixas de diálogo em múltiplas páginas são úteis, pois elas permitem que você apresente informações em pequenas e organizadas porções. A caixa de diálogo Formatar Células do Excel (que é exibida quando você clica com o botão direito em uma célula e escolhe Formatar Células) é um bom ExcelVBABokks.indb 308

15/02/2013 14:00:16



Capítulo 18: Técnicas e Truques do UserForm

ajudar a organizar algumas opções de exibição do Excel.

Criar as suas próprias caixas de diálogo multi-páginas é relativamente fácil, graças ao controle Multipáginas. A Figura 18-11 mostra uma caixa de diálogo personalizada que usa um controle Multipágina com três páginas. Quando o usuário clicar uma página, a nova página é ativada e apenas os controles a ela relacionados são exibidos.

Observe que esta é uma caixa de diálogo modeless. Em outras palavras, o usuário pode mantê-la exibida enquanto estiver trabalhando. Cada um dos controles tem um efeito imediato, portanto, não é necessário ter um botão OK.

Figura 18-11:

As três
páginas de
um controle
Multipágina.

Tenha em mente os seguintes pontos ao usar o controle Multipáginas
BRE-SE

para criar uma caixa de diálogo:

LEM

5 Use apenas um controle Multipáginas por caixa de diálogo.

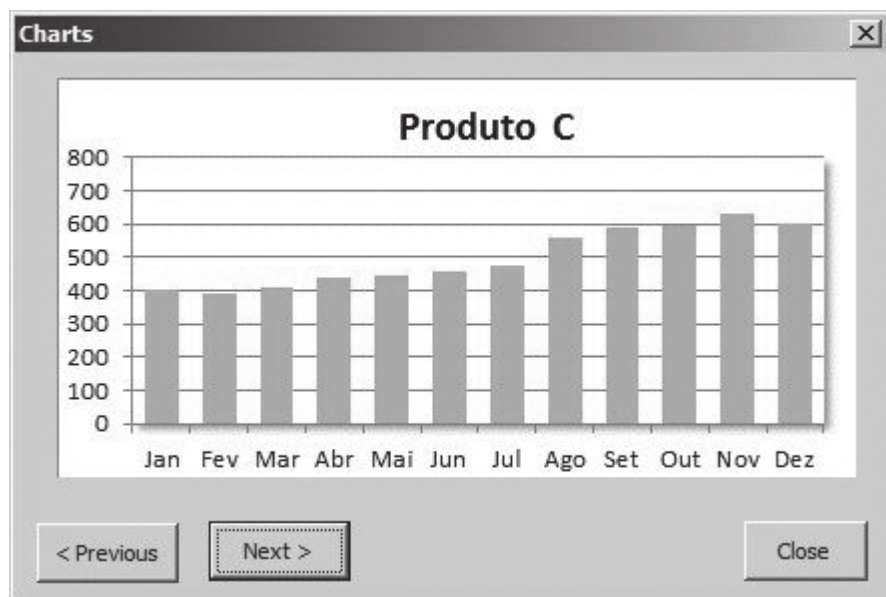
5 Assegure-se de usar o controle Multipágina, não o controle TabStrip. O controle TabStrip é mais difícil de usar.

5 Para tornar alguns controles (tais como os botões OK, Cancelar ou Fechar) visíveis o tempo todo, coloque esses controles fora do controle Multipágina.

5 Clique com o botão direito em uma página no controle para exibir um menu de atalho que permite acrescentar, remover, renomear ou mover a página.

ExcelVBABokks.indb 309

15/02/2013 14:00:17



310 Parte IV: Como se Comunicar com Seus Usuários

5 Por ocasião do projeto, clique em uma página para ativá-la.

Depois que ela estiver ativada, adicione outros controles à página usando os procedimentos normais.

5 Para selecionar o próprio controle Multipágina (ao invés de uma página no controle), clique a margem do controle. Fique de olho na janela Propriedades, que exibe o nome e o tipo do controle selecionado. Você também pode selecionar o controle

selecionando o seu nome a partir da lista drop-down na janela Propriedades.

5 Você pode mudar a aparência do controle, alterando as propriedades Style e TabOrientation.

5 A propriedade Value de um controle Multipáginas determina qual página é exibida. Por exemplo, se você escrever um código e configurar a propriedade Value para 0, a primeira página do controle é exibida.

Este exemplo está disponível no site deste livro.

Exibindo um gráfico em um UserForm

Se você precisa exibir um gráfico em um UserForm, descubra que o Excel não oferece qualquer maneira direta de fazê-lo. Portanto, você precisa ser criativo. Esta seção descreve uma técnica para permitir que você exiba um ou mais gráficos em um UserForm.

A Figura 18-12 mostra um exemplo, que exibe três gráficos. O UserForm tem um controle Imagem. O truque é usar código VBA para salvar o gráfico como um arquivo GIF (Graphic User Interface) e, depois, especificar qual arquivo tem a propriedade Picture do controle Imagem (que carrega a imagem do seu disco). Os botões Previous e Next trocam o gráfico exibido.

Figura 18-12:

Exibindo

um gráfico

em um

UserForm.

ExcelVBABokks.indb 310

15/02/2013 14:00:17

Neste exemplo, que também está disponível no site deste livro, os três gráficos estão em uma planilha chamada Charts. Os botões Previous e Next determinam qual gráfico exibir, e o número desse gráfico é armazenado como uma variável Public chamada ChartNum, que é acessível a todos os procedimentos. Um procedimento chamado UpdateChart, que é listado aqui, faz o verdadeiro trabalho.

```
Private Sub UpdateChart()
```

```
Dim CurrentChart As Chart
```

```
Dim Fname As String
```

```
Set CurrentChart = _
```

```
Sheets("Charts").ChartObjects(ChartNum).Chart
```

```
CurrentChart.Parent.Width = 300
```

```
CurrentChart.Parent.Height = 150
```

```
‘ Salva o gráfico como GIF
```

```
Fname = ThisWorkbook.Path & "\temp.gif"
```

```
CurrentChart.Export FileName: = Fname, _
```

```
FilterName: = "GIF"
```

```
‘ Mostra o gráfico
```

Image1.Picture = LoadPicture(Fname)

End Sub

Este procedimento determina um nome para o gráfico salvo e depois, usa o método Export para exportar o arquivo GIF. Finalmente, ele usa a função VBA LoadPicture para especificar a propriedade Picture do objeto Image.

Uma Lista de Verificação

de Caixa de Diálogo

Eu concluo este capítulo com uma lista de verificação para usar ao criar caixas de diálogo:

- ☐ Os controles estão alinhados entre si?
- ☐ Os controles similares têm o mesmo tamanho?
- ☐ Os controles estão igualmente espaçados?
- ☐ A caixa de diálogo tem uma legenda adequada?
- ☐ A caixa de diálogo está muito pesada? Se estiver, você pode querer usar uma série de caixas de diálogo ou dividi-las por um controle Multipágina.
- ☐ O usuário pode acessar cada controle com uma tecla de atalho?
- ☐ Há teclas de atalho repetidas?
- ☐ Os controles estão agrupados logicamente, por função?

ExcelVBABokks.indb 311

15/02/2013 14:00:17

312 Parte IV: Como se Comunicar com Seus Usuários

- ☐ A ordem de tabulação está configurada corretamente? O usuário deve ser capaz de tabular através da caixa de diálogo e acessar controles sequencialmente.
- ☐ Se você planeja armazenar a caixa de diálogo em um add-in, você a

testou cuidadosamente depois de criar o add-in?

- ☐ O seu código VBA agirá de forma apropriada se o usuário cancelar a caixa de diálogo, pressionar Esc ou usar o botão de fechar?
- ☐ O texto contém qualquer erro de ortografia? Infelizmente, o corretor do Excel não funciona com UserForms, assim, você está por sua conta quando se trata de escrever corretamente.
- ☐ A sua caixa de diálogo caberá na tela na resolução mais baixa que for usada (normalmente, no modo 800x600)? Em outras palavras, se você desenvolver sua caixa de diálogo usando um modo de vídeo de alta resolução, a sua caixa de diálogo pode ser grande demais para caber em uma tela de resolução mais baixa.
- ☐ Todas as caixas de texto têm a configuração apropriada de validação? Se você pretende usar a propriedade WordWrap, a propriedade MultiLane também está configurada para True?
- ☐ Todas as barras de rolagem e botões de rotação permitem apenas valores válidos?
- ☐ Todos as caixas de listagem têm sua propriedade MultiSelect configurada adequadamente?

A melhor maneira de administrar caixas de diálogo personalizadas é criar caixas de diálogo – muitas delas. Comece simplesmente e experimente com os controles e suas propriedades. E não se esqueça do sistema de ajuda: ele é a sua melhor fonte de detalhes sobre cada controle e propriedade.

ExcelVBABokks.indb 312

15/02/2013 14:00:17

Capítulo 19

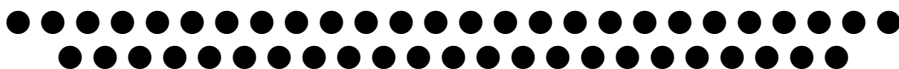
Como Acessar suas Macros

através da Interface

de Usuário



- Uma espiada na personalização da Faixa de Opções usando XML
- Adicionando itens a um menu de clicar com o botão direito
- Como adicionar um botão à barra de ferramentas de acesso rápido (manualmente)
- Personalizando a Faixa de Opções (manualmente)



Antes do Excel 2007, os usuários tinham acesso a dezenas de barras de ferramentas integradas, e a criação de novas barras de ferramentas foi repentina. Porém, como escreveu Bob Dylan, “Os tempos estão mudando”. Este capítulo descreve o que mudou, começando com o Excel 2007, e mostra um pouco do que você pode fazer para exibir suas macros na interface de usuário.

O Que Aconteceu com CommandBars?

Ao programar em Excel 2003 e antes, você escrevia código para criar uma barra de ferramentas (chamada de CommandBar em VBA). A barra de ferramentas continha botões, para permitir ao usuário (ou a você mesmo) acessar suas macros. Além disso, você podia usar o objeto CommandBar para adicionar novos comandos aos menus do Excel. A partir do Excel 2007, é a nova interface de usuário, a Faixa de Opções (Ribbon), que muda drasticamente o quadro. São boas e más notícias.

A boa notícia é que a maioria do antigo código de CommandBar escrito em VBA ainda funcionará.

A má notícia é que o seu código VBA bem refinado, que tenta criar uma barra de ferramentas ou acrescentar um comando a um menu é interceptado pelo Excel. Ao invés de exibir o aperfeiçoamento bem pensado de sua interface, o Excel 2007 (e o Excel 2010) simplesmente move os seus menus e barras de ferramentas personalizadas para um tabulador da Faixa de Opções que engloba tudo, chamado Add-Ins.

Quase tudo o que escrevi neste livro aplica-se tanto ao Excel 2007 quanto ao Excel 2010. Estou para apresentar uma diferença significativa entre essas duas versões, e ela envolve personalizar a Faixa de opções.

Se você usa o Excel 2007, tem uma maneira de personalizar a Faixa de opções: aprenda a escrever código RibbonX e adicione a modificação a uma pasta de trabalho. Essa não é uma tarefa simples, como verá mais adiante. Mas, se usar o Excel 2010, você também pode modificar a Faixa de opções manualmente, usando a guia Personalizar Faixa de Opções da caixa de diálogo Opções do Excel.

Você não pode fazer alterações ao Ribbon usando VBA. Triste, mas é **BRE-SE**

verdade. Por exemplo, se você escrever um aplicativo e quiser acres-

centar alguns novos botões ao Ribbon, precisa de um programa que faça mudanças fora do Excel, usando algo chamado RibbonX.

Como personalizar manualmente a Faixa

de Opções

É fácil fazer alterações manualmente à Faixa de Opções, mas você deve usar o Excel 2010. Se usar Excel 2007, simplesmente pule esta seção, porque ela não se aplica a você.

É possível personalizar a Faixa de Opções destas maneiras:

5 Guias

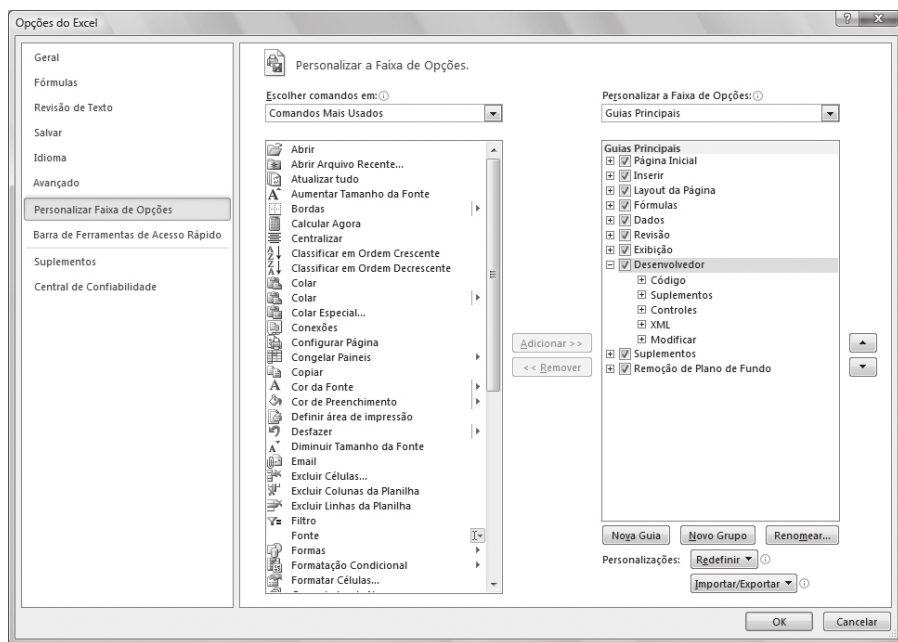
- Adicione uma nova guia personalizada.
- Apague guias personalizadas.
- Adicione um novo grupo à guia.
- Mude a ordem das guias.
- Mude o nome de uma guia.
- Oculte guias integradas.

5 Grupos

- Adicione novos grupos personalizados.
- Adicione comandos a um grupo personalizado.
- Remova comandos dos grupos personalizados.

ExcelVBABokks.indb 314

15/02/2013 14:00:17



Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 315

- Remova grupos de uma guia.
- Mova um grupo para uma guia diferente.
- Mude a ordem dos grupos dentro de uma guia.
- Mude o nome de um grupo.

Esta é uma lista bem compreensível de opções de personalização, mas há algumas ações que você *não pode* fazer (não importa quanto tente):

5 Você não pode remover guias integradas - mas *pode* ocultá-las.

5 Você não pode remover comandos de grupos integrados.

5 Você não pode mudar a ordem de comandos em um grupo integrado.

Você faz mudanças manuais à Faixa de Opções no painel Personalizar Faixa de Opções da caixa de diálogo Opções do Excel (veja a Figura 19-1). A maneira mais rápida de exibir esta caixa de diálogo é clicar com o botão direito em qualquer lugar na Faixa de Opções e escolher Personalizar Faixa de Opções.

Figura 19-1:

A guia

Personalizar

Faixa de

Opções da

caixa de

diálogo

Opções do

Excel.

O processo de personalizar a Faixa de Opções é muito similar a personalizar a barra de ferramentas de Acesso Rápido, que é descrito mais adiante neste capítulo. A única diferença é que você precisa decidir se coloca o comando dentro da Faixa de Opções. O procedimento geral é: ExcelVBABokks.indb 315

15/02/2013 14:00:17

316 Parte IV: Como se Comunicar com Seus Usuários 1. Use a lista drop-down à esquerda (chamada como Escolher comandos em) para exibir vários grupos de comandos.

2. Localize o comando na caixa de listagem e selecione-o.
3. Use a lista drop-down à direita (chamada como Personalizar a Faixa de Opções) para escolher um grupo de guias. Guias Principais refere-se às guias que estão sempre visíveis; Guias de Ferramenta refere-se às guias de contexto que aparecem quando um objeto em especial é selecionado.
4. Na caixa de listagem à direita, selecione a guia e o grupo onde você gostaria de colocar o comando. Você precisará clicar nos controles e no “sinal de adição” para expandir as listas hierárquicas.
5. Clique no botão Adicionar para adicionar o comando selecionado da esquerda para o grupo à direita.

Tenha em mente que você pode usar o botão Nova Guia para criar uma nova guia, e o botão Novo Grupo para criar um novo grupo dentro de uma guia. As novas guias e grupos recebem nomes genéricos, portanto, possivelmente você irá querer dar a eles nomes mais significativos. Use o botão Renomear para renomear a guia ou grupo selecionado. Também é possível renomear guias e grupos integrados.

BRE-SE

Embora você não possa remover uma guia integrada, pode ocultar a **LEM**

guia, desmarcando a caixa de verificação próxima ao seu nome.

Por sorte, também é possível acrescentar macros à Faixa de Opções.

Escolha Macros a partir da lista drop-down à esquerda, e todas as macros atualmente disponíveis são relacionadas, prontas para serem adicionadas à Faixa de Opções.

Se você personalizar a Faixa de Opções para incluir uma macro, o **BRE-SE**

comando macro fica visível na Faixa de Opções, mesmo quando a pasta **LEM**

de trabalho que contém a macro não estiver aberta. Clicar o comando abrirá a pasta de trabalho que contém a macro.

Personalizando a Faixa de Opções com XML

Em algumas situações, você pode querer modificar automaticamente a Faixa de Opções, quando uma pasta de trabalho ou add-in estiver aberto. Fazer isso facilita ao usuário acessar a sua macro. Isso também elimina a necessidade do usuário de modificar manualmente a Faixa de Opções, usando a caixa de diálogo Opções do Excel.

Fazer alterações automáticas na Faixa de Opções pode ser feito com o Excel 2007 e o Excel 2010, mas não é fácil. Modificar a Faixa de Opções envolve escrever código XML (Extensible Markup Language) em um editor de texto, copiar aquele arquivo XML no arquivo da pasta de trabalho, editar um punhado de arquivos XML (que também são afastados dentro do novo formato de arquivo do Excel, o que, de fato, nada mais é do que um contêiner compactado de arquivos individuais

— mas relacionados) e, depois, escrever procedimentos VBA para lidar com o clique dos controles que você põe no arquivo XML.

ExcelVBABokks.indb 316

15/02/2013 14:00:18

Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 317

Obtenha o software

Se você quiser acompanhar o exemplo de personalização da Faixa de Opções, precisa fazer o download de um pequeno programa chamado Custom UI Editor (Editor de Interface de Usuário Personalizada) para o Microsoft Office. Esse é um programa gratuito que simplifica enormemen-te o processo de personalizar a Faixa de Opções nos aplicativos do Microsoft Office. Usar esse software ainda requer muito trabalho, mas é muito mais fácil do que fazer manualmente Enquanto eu escrevia isto, o software estava disponível aqui: <http://openxmldeveloper.org/articles/customuieditor.aspx> Se esta URL não funcionar, busque na Web por “Custom UI Editor for Microsoft Office” e você encontrará o software. Ele tem um pequeno download e é gratuito.

Felizmente, o software está disponível para ajudá-lo com a personalização da Faixa de Opções — mas, você ainda precisa conhecer termos com XML.

Explicar todos os detalhes intrínsecos envolvidos em personalizar a Faixa de Opções está bem além do escopo deste livro. Entretanto, eu o

encaminho através de um rápido exemplo que demonstra as etapas exigidas para acrescentar um novo grupo à guia Página Inicial. O novo grupo é chamado de Excel VBA Para Leigos e contém um botão Click Me. Clicar tal botão roda uma macro VBA chamada ShowMessage.

Você pode fazer o download desse exemplo a partir do site deste livro, que contém esta personalização. Se você quiser criá-la sozinho, siga exatamente estas etapas:

- 1. Crie uma nova pasta de trabalho Excel.**
- 2. Salve a pasta de trabalho e nomeie como ribbon modification.xlsm.**
- 3. Feche a pasta de trabalho. Esta etapa é muito importante.**
- 4. Inicie o Custom UI Editor do Microsoft Office. Se não tiver esse software, você precisa encontrá-lo e instalá-lo. Consulte o tópico “Obtenha o software”, anteriormente neste capítulo.**
- 5. No Custom UI Editor, escolha File⇒Open e encontre a pasta de trabalho que você salvou na Etapa 2.**
- 6. Escolha Insert⇒Office 2007 Custom UI Part. Escolha o respectivo comando se você estiver usando Excel 2010.**
- 7. Digite o seguinte código no painel de código (chamado customUI.xml), exibido no Custom UI Editor (veja a Figura 19-2):**
ExcelVBABokks.indb 317

15/02/2013 14:00:18

318 Parte IV: Como se Comunicar com Seus Usuários

```
< customUI xmlns = 'http://schemas.microsoft.com/  
office/2006/01/customui' >
```

```
< ribbon >
```

```
< tabs >
```

```
< tab idMso = 'TabHome' >
```

```
< group id = 'Group1' label = 'Excel VBA For Dummies' >
```



```
< button  
id = 'Button1'  
label = 'Click  
Me'  
size = 'large'  
onAction = 'ShowMessage'  
imageMso = 'FileStartWorkflow'  
/>  
  
</group>  
  
</tab>  
  
</tabs>  
  
</ribbon>  
  
</customUI>
```

8. Clique o botão Validate na barra de ferramentas. Se o código tiver quaisquer erros de sintaxe, você receberá uma mensagem que descreve o problema. Se quaisquer erros forem identificados, você precisa corrigi-los.

9. Clique o botão Generate Callback.

CustonUI Editor cria um procedimento VBA Sub que é executado

quando o botão é clicado (veja a Figura 19-3). Na verdade, esse procedimento não é inserido na pasta de trabalho, assim, você precisa copiá-lo para uso posterior (ou memorizá-lo, se tiver uma boa memória).

10. Volte para o módulo customUI.xml e escolha File⇨Save (ou clique o ícone Save na barra de ferramentas).

11. Feche o arquivo usando o comando File⇨Close.

12. Abra a pasta de trabalho no Excel. Clique na Guia Página Inicial e você deverá ver um novo grupo na Faixa de Opções e um

botão Ribbon. Mas, ainda não funciona.

13. Pressione Alt + F11 para ativar o VBE.

14. Insira um novo módulo VBA e cole (ou digite) o procedimento que foi gerado na Etapa 9. Acrescente uma declaração MsgBox, assim você saberá se o procedimento está, de fato, sendo executado. O procedimento é:

```
Sub ShowMessage(control As IRibbonControl)
```

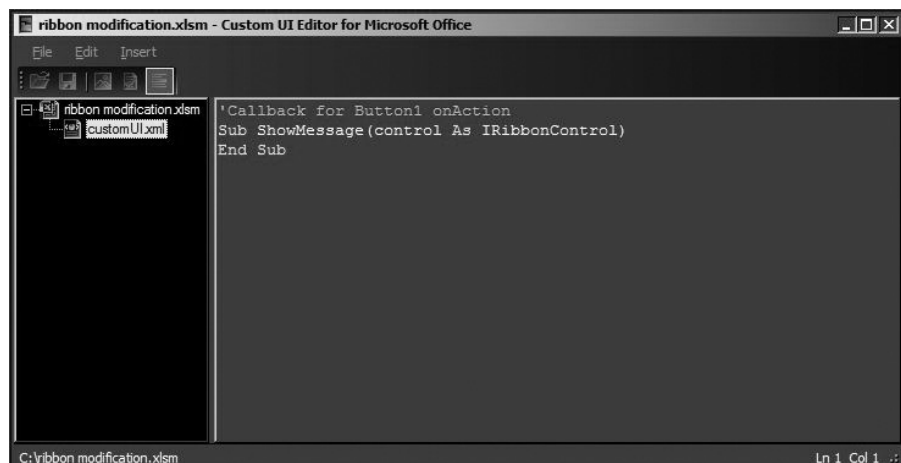
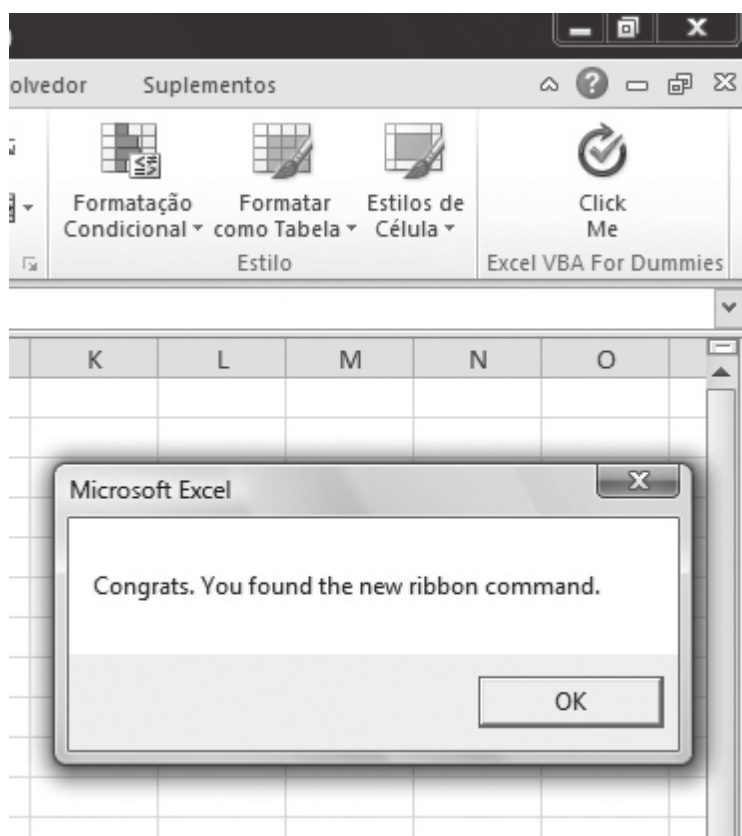
```
MsgBox "Congrats. You found the new ribbon  
command."
```

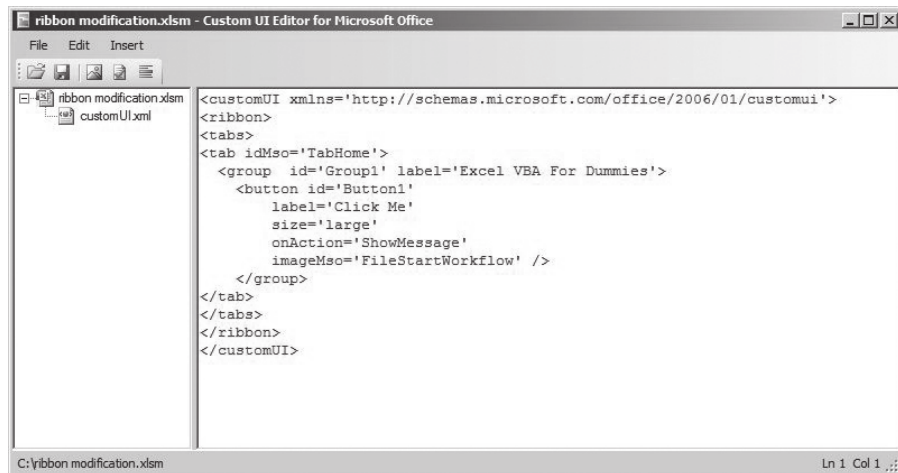
```
End Sub
```

15. Pressione Alt + F11 para pular de volta para o Excel. Clique o novo botão da Faixa de Opções. Se tudo correu bem, você verá a MsgBox mostrada na Figura 19-4.

ExcelVBABokks.indb 318

15/02/2013 14:00:18





Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 319

Figura 19-2:

Código

RibbonX

exibido no

Custom UI

Editor.

Figura 19-3:

O procedimento VBA

de teste que

é executado

clicando no

botão da

Faixa de

Opções.

Figura 19-4:

Prova de
que, de fato,
é possível
adicionar um
comando na
Faixa de
Opções
usando XML.

ExcelVBABokks.indb 319

15/02/2013 14:00:18

320 Parte IV: Como se Comunicar com Seus Usuários No Custom UI Editor, quando você escolhe Inserir>Office 2007 Custom UI **BRE-SE**

Part, você insere uma parte UI (Interface de Usuário) do Excel 2007. O

LEM

Custom UI Editor também tem essa opção para o Excel 2010. Se escolher essa opção, a modificação da Faixa de Opções não funcionará no Excel 2007. Além disso, você precisa especificar um Namespace diferente na primeira declaração XML: <customUI xmlns='http://schemas.microsoft.com/

office/2009/07/customUI'>

Você só precisa inserir uma parte UI no Excel 2010 se usar recursos que são específicos para o Excel 2010. Também é possível ter uma parte UI para cada versão, mas isso raramente é necessário.

Provavelmente, você notou que fazer modificações na Faixa de Opções usando XML não é exatamente intuitivo. Mesmo com uma boa ferramenta de ajuda (tal como o Custom UI Editor), você ainda precisa entender XML. Se isso parece atraente, verifique as livrarias. Você encontrará livros dedicados exclusivamente à personalização da interface da Faixa de Opções do Microsoft Office. Este não é um deles.

Acrescentando um botão à barra de

ferramentas de Acesso Rápido

Se você criar uma macro que usa com frequência, Se você usa o Excel 2007, o botão da barra de pode querer acrescentar um novo botão à barra ferramentas de Acesso Rápido só funciona de ferramentas de Acesso Rápido. Fazer isso é quando a pasta de trabalho que contém a ma-fácil, mas você deve fazê-lo manualmente. A barra cro estiver aberta. Pior ainda, a macro só funde ferramentas de Acesso Rápido destina-se a ser ciona quando aquela pasta de trabalho for personalizada apenas pelos usuários finais — não uma pasta de trabalho ativa. Porém, esses programadores. Eis como fazer: problemas foram corrigidos no Excel 2010.

Quando você clica um botão de macro na bar-

1. Clique com o botão direito na barra de ferra-

ra de ferramentas de Acesso Rápido, a pasta de

mentas de Acesso Rápido e selecione trabalho que contém a macro é aberta (se já **Personalizar Barra de Ferramentas de** não estiver aberta). E a macro pode ser execu-Acesso Rápido para exibir a respectiva guia tada quando qualquer pasta de trabalho estina caixa de diálogo Options (Opções) do ver aberta.

Excel. No Excel 2007, essa guia é chamada

de Personalizar.

O Excel 2010 também tem uma opção para exi-

2. Na lista drop-down chamada como Escolher bir o botão da barra de ferramentas de Acesso **comandos em, selecione Macros.**

Rápido apenas quando uma pasta de trabalho

em especial estiver aberta. Antes de você

3. Selecione da lista a sua macro.

acrescentar a macro, use a lista drop-down à

4. Clique o botão Adicionar e a macro é acres-

direita da caixa de diálogo Opções do Excel, e

centada à lista da barra de ferramentas de especifique o nome da

pasta de trabalho, ao **Acesso Rápido, à direita.**

invés de Para todos Documentos (Padrão).

5. Se você quiser, clique o botão Modificar Se você tiver macros que são usadas em muitas **para mudar o ícone e (opcionalmente) o** pastas de trabalho diferentes, uma boa ideia é **nome a exibir.**

armazená-las em sua Personal Macro Workbook.

ExcelVBABokks.indb 320

15/02/2013 14:00:18

Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 321

Como essa coisa de XML é complexa demais para o programador VBA iniciante, o restante deste capítulo aborda a personalização da UI (Interface de Usuário) usando o *antigo* método (apenas VBA). Ele não é tão esperto quanto a Faixa de Opções, mas é muito mais fácil e ainda oferece acesso rápido às suas macros.

Personalizando Menus de Atalho

Antes do Excel 2007, os programadores de VBA usavam o objeto

CommandBar para criar menus personalizados, barras de ferramentas personalizadas e menus de atalho (a partir do botão direito) personalizadas.

Começando com o Excel 2007, o objeto CommandBar está em uma posição bastante estranha. Se você escrever código para personalizar um menu ou uma barra de ferramentas, o Excel intercepta aquele código e ignora muitos de seus comandos. Como observei anteriormente neste capítulo, as personalizações de menu e barra de ferramentas acabam em Add-Ins\Menu Commands ou no grupo Add-Ins\Custom Toolbars.

Assim, em todos os objetivos práticos, você está limitado aos menus de atalho. E é isso que discuto nas seguintes seções.

Comandando a coleção de CommandBars

O Excel suporta três tipos de CommandBars, diferenciados pela sua propriedade Type. Para os menus de atalho, você está interessado no

tipo 2, também conhecido pelo seu nome enquanto constante integrada, `msoBarTypePopup`.

Listando todos os menus de atalho

O procedimento relacionado aqui usa a coleção `CommandBars`. Ele exibe, em uma planilha, os nomes de todos os menus de atalho —

`CommandBars` que têm uma propriedade `Type` de 2 (`msoBarTypePopup`). Para cada `CommandBar`, o procedimento lista o seu índice e nome.

```
Sub ShowShortcutMenusName()
```

```
    Dim Row As Long
```

```
    Dim cbar As CommandBar
```

```
    Row = 1
```

```
    For Each cbar In Application.commandBars
```

```
        If cbar.Type = msoBarTypePopup then
```

```
            Cells(Row, 1) = cbar.Index
```

```
            Cells(Row, 2) = cbar.Name
```


Row = Row + 1

End If

Next cbar

End Sub

ExcelVBABokks.indb 321

15/02/2013 14:00:19

	A	B
2	22	PivotChart Menu
3	35	Workbook tabs
4	36	Cell
5	37	Column
6	38	Row
7	39	Cell
8	40	Column
9	41	Row
10	42	Ply
11	43	XLM Cell
12	44	Document
13	45	Desktop
14	46	Nondefault Drag and Drop
15	47	AutoFill
16	48	Button
17	49	Dialog

322 Parte IV: Como se Comunicar com Seus Usuários

A Figura 19-5 mostra uma parte do resultado obtido ao executar esse procedimento, que está disponível no Web site deste livro. Por exemplo, você vê que o menu de atalho chamado Workbook Tabs tem um índice de 34. Este é o menu de atalho que aparece quando você

clica com o botão direito sobre uma guia da planilha.

Figura 19-5:

Uma macro

VBA produziu

esta lista de

todos os

nomes de

menus de

atalho.

Referência a CommandBars

Você pode se referir a um CommandBar em especial pelo seu Índice ou pelo seu nome. Se olhar a Figura 19-5, verá que o menu acessível pelo botão direito Cell tem um Índice de 35 ou 38. Isso porque o conteúdo desse menu difere quando o Excel está em uma posição diferente. O

número 35 é o que você obtém quando o Excel está em seu modo de visualização Normal, o número 38 mostra quando você está no modo Visualização de Quebra de Página. Você pode fazer referência ao menu de atalho em qualquer de duas maneiras:

`Application.CommandBars(35)`

ou

`Application.commandBars("Cell")`

Ainda que haja duas Células CommandBars, a segunda linha de código **CUIDADO!**

acima sempre encaminha aquela com o índice 35. Por algum motivo, a Microsoft não é consistente na nomeação de CommandBars. Você

poderia esperar que cada CommandBar tivesse o seu próprio nome, mas obviamente, eles não têm. Os menus acessíveis pelo botão direito que ExcelVBABokks.indb 322



Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 323

diferem em conteúdo — dependendo da posição em que o Excel está — aparecem mais de uma vez na lista de CommandBars disponíveis.

Você poderia pensar que é melhor fazer referência a CommandBars usando sua propriedade Index. Errado! Mas, até isso pode causar problemas, porque os números do Index nem sempre permaneceram constantes através das diferentes versões do Excel. Na verdade, a propriedade Index pode até variar dentro de uma única versão do Excel.

Referência a controles em um CommandBar

Um objeto CommandBar contém controles, que são botões, menus ou itens de menu. O seguinte procedimento exibe a propriedade Caption para o primeiro controle no menu de célula acessível pelo botão direito: Sub ShowCaption()

```
MsgBox Application.CommandBars("Cell"). _
```

```
Controls(1).Caption
```

```
End Sub
```

Quando você executa este procedimento, vê a caixa de mensagem mostrada na Figura 19-6. O e comercial (&) é usado para indicar a letra sublinhada no texto — o toque de teclado que executará o item do menu.

Figura 19-6:

Exibindo a
propriedade

Caption em

um controle.

Em alguns casos, objetos Control em um menu de atalho contêm outros **BRE-SE**

objetos Control. Por exemplo, o controle Sort, obtido ao clicar com o botão **LEM**

direito nas células contêm outros controles.

Cada controle tem uma propriedade Name e uma Id. Você pode acessar um controle usando qualquer dessas propriedades (mas localizar um controle pela sua Id é um pouco mais complexo): Sub AccessControlByName()

```
MsgBox CommandBars("Cell").Controls("Copy").Caption
```

```
End Sub
```

```
Sub AccessControlById()
```

```
MsgBox CommandBars("Cell").FindControl(ID:=19).Caption
```

```
End Sub
```

ExcelVBABokks.indb 323

15/02/2013 14:00:19

324 Parte IV: Como se Comunicar com Seus Usuários Não use a propriedade Caption para acessar um controle se estiver escrevendo código que pode ser utilizado por usuários com uma versão **CUIDADO!**

de linguagem diferente de Excel. Caption é uma linguagem específica, portanto, o seu código falhará nos sistemas daqueles usuários. Ao invés disso use o método FindControl junto com a Id do controle (que independe de linguagem). Felizmente, os nomes de CommandBar não são internacionalizados.

Propriedades de controles CommandBar

Os controles CommandBar têm uma série de propriedades que determinam aspectos como a aparência e o funcionamento dos controles. Esta lista contém algumas das propriedades mais úteis dos controles CommandBars: 5 **Caption:** O texto exibido para o controle. Se o controle só mostrar uma imagem, Caption aparece quando você move o mouse sobre o controle.

5 **FaceID:** Um número que representa uma imagem gráfica exibida próxima ao texto do controle.

5 **BeginGroup:** True se aparecer uma barra de separação antes do controle.

5 **OnAction:** O nome de uma macro VBA que executa quando o usuário clica o controle.

5 **BuiltIn:** True se o controle for um controle integrado do Excel.

5 **Enabled (habilitado):** True se o controle puder ser clicado.

5 **Visible:** True se o controle for visível. Muitos dos menus de atalho contêm controles ocultos.

5 **ToolTipText:** Texto que aparece quando o usuário move o cursor do mouse sobre o controle.

O procedimento ShowShortcutMenuItems lista todos os controles de primeiro nível em cada menu de atalho. Além disso, ele identifica controles ocultos, colocando suas Legendas entre colchetes.

Sub ShowShortcutMenuItems()

Dim Row As Long, Col As Long

Dim Cbar As CommandBar

Dim Ctl As CommandBarControl

Row = 1

Application.ScreenUpdating = False

For Each Cbar In Application.CommandBars

If Cbar.Type = msoBarTypePopup then

Cells(Row, 1) = Cbar.Index

Cells(Row, 2) = Cbar.Name

Col

=

3

For Each Ctl In Cbar.Controls

ExcelVBABokks.indb 324

15/02/2013 14:00:19

	A	B	C	D	E	F	G
1	22	PivotChart Menu	&Configurações de Campo	&Opções...	Atualizar Dados	Ocultar & botões de can	Fórmulas
2	35	Workbook tabs	Sheet1	<&Lista de planilhas>	<&Lista de planilhas>	<&Lista de planilhas>	<&Lista de planilhas>
3	36	Cell	R&ecortar	Copiar	Colar	&Colar Especial...	&Colar tabela
4	37	Column	R&ecortar	Copiar	Colar	&Colar Especial...	&Colar tabela
5	38	Row	R&ecortar	Copiar	Colar	&Colar Especial...	&Colar tabela
6	39	Cell	R&ecortar	Copiar	Colar	&Colar Especial...	&Colar tabela
7	40	Column	R&ecortar	Copiar	Colar	&Colar Especial...	&Colar tabela
8	41	Row	R&ecortar	Copiar	Colar	&Colar Especial...	&Colar tabela
9	42	Ply	&Inserir...	&Excluir	Re&nomear	&Mover ou copiar...	E&xibir Código
10	43	XLM Cell	R&ecortar	Copiar	Colar	&Colar Especial...	&Colar tabela
11	44	Document	&Salvar	Salvar & como...	&Imprimir...	Config&urar Página...	&Verificar Ortogra
12	45	Desktop	&Novo...	&Abrir...	Salvar Espaço de Trabalho...	&Calcular Agora	<Tela Intei&ra>
13	46	Nondefault Drag and Drop	&Mover aqui	Copiar & aqui	Copiar aqui & somente como va	Copiar aqui somente co	Criar & vínculo aqui
14	47	AutoFill	&Copiar células	&Preencher Série	Preencher & formatação somen	Preencher & sem forma	Preencher & dias
15	48	Button	R&ecortar	Copiar	Copiar Tinta como Te&xto	Colar	Limpar
16	49	Dialog	Colar	Ordem de Ta&bulação...	&Executar Caixa de Diálogo	<Tela Intei&ra>	
17	50	Series	Objeto &selecionado	&Tipo de gráfico...	Selecionar Dados...	Ad&icionar linha de ter	Li&mpar
18	51	Plot Area	Objeto &selecionado	&Tipo de gráfico...	Selecionar Dados...	O&pções de gráfico...	&Mover Gráfico...
19	52	Floor and Walls	Objeto &selecionado	Exibição &3D...	Li&mpar		
20	53	Trendline	Objeto &selecionado	Li&mpar			
21	54	Chart	Objeto &selecionado	Li&mpar			
22	55	Format Data Series	Objeto &selecionado	&Tipo de gráfico...	Selecionar Dados...	Ad&icionar linha de ter	&Ocultar Detalhe
23	56	Format Axis	Objeto &selecionado	Li&mpar	&Ocultar Detalhe	Mostrar Detal&he	
24	57	Format Legend Entry	Objeto &selecionado	&Ocultar Detalhe	Mostrar Detal&he	Li&mpar	
25	58	Formula Bar	R&ecortar	Copiar	Colar	&Formatar células...	Escolher na Lista S
26	59	PivotTable Context Menu	Copiar	&Formatar células...	<For&mato de Número...>	Atualiza&r	<&Classificar>

If Ctl.Visible Then

Cells(Row,

Col) =

Ctl.Caption

Else

Cells(Row, Col) = “<” & Ctl.Caption & “>”

End

If

Col = Col + 1

Next

Ctl

Row = Row + 1

End If

Next Cbar

End Sub

A Figura 19-7 mostra uma parte da saída.

O procedimento ShowShortcutMenuItems está disponível no site deste livro. Se você rodar a macro, poderá ver que muitos dos menus de atalho contém controles ocultos.

Figura 19-7:

Listagem de

todos os

controles de

alto nível em

todos os

menus de

atalho.

Esta foi uma visão geral rápida de CommandBars. Claro que há muito mais sobre CommandBars, mas é o máximo em que posso ir neste livro.

A próxima seção oferece alguns exemplos que podem ajudar a esclarecer qualquer confusão que você tenha feito.

Com a introdução da nova interface de usuário, a Faixa de Opções muita coisa mudou. Algumas das mudanças são para melhor e algumas são para pior. As possibilidades de conseguir controle sobre a interface de usuário usando apenas VBA agora são muito limitadas.

ExcelVBABokks.indb 325

15/02/2013 14:00:20

326 Parte IV: Como se Comunicar com Seus Usuários *Exemplos de Menu de Atalho VBA*

Esta seção contém alguns exemplos do uso de VBA para manipular os menus obtidos com o botão direito do mouse — normalmente conhecidos como menus de atalho. Esses exemplos dão uma ideia dos tipos de coisas que você pode fazer e todos eles podem ser modificados para se adequar às suas necessidades.

Reconfigure todos os menus integrados de clicar com o botão direito.
O

seguinte procedimento reconfigura todas as barras de ferramentas integradas à sua posição original:

```
Sub ResetAll()
```

```
Dim cbar As CommandBar
```

```
For Each cbar In Application.CommandBars
```



```
If cbar.Type = msoBarTypePopup Then
```

```
cbar.Reset
```

```
cbar.Enabled = True
```

```
End If
```

```
Next cbar
```

```
End Sub
```

Este procedimento não terá efeito a menos que alguém tenha executado algum código VBA que adicione itens, remova itens ou desative menus de atalho.

*Adicionando um novo item ao menu de
atalho Cell*

No Capítulo 16, descrevi o utilitário Change Case. Você pode aperfeiçoar um pouco esse utilitário, tornando-o disponível a partir do menu de atalho da célula.

Este exemplo está disponível no site deste livro.

O procedimento AddToShortcut acrescenta um novo item ao menu de atalho. Lembre-se que o Excel tem dois menus de atalho para células.

Este procedimento modifica o menu normal, mas não o menu que aparece no modo Visualização de Quebra de Página.

```
Sub AddToShortcut()
```

Dim Bar As CommandBar

Dim NewControl As CommandBarButton

DeleteFromShortcut

Set Bar = Application.CommandBars("Cell")

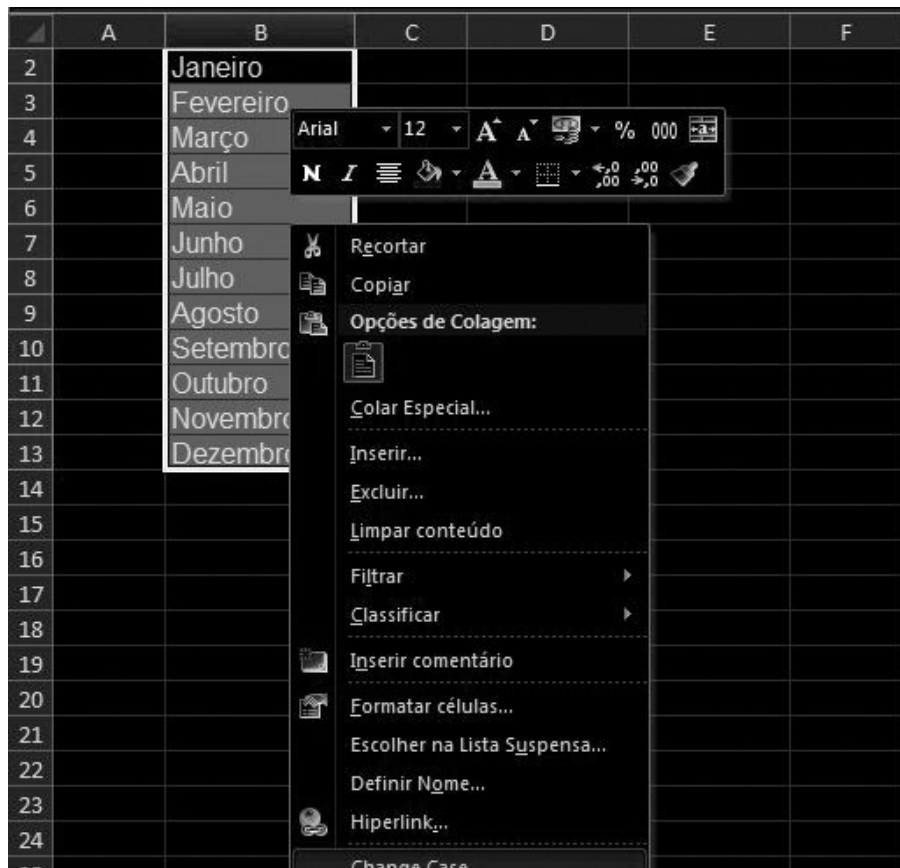
(Type:=msoControlButton, ID:=1, _
temporary:=True)

With NewControl

.Caption = "&Change Case"

ExcelVBABokks.indb 326

15/02/2013 14:00:20



Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 327

.OnAction = "ChangeCase"

.Style = msoButtonIconAndCaption

End With

End Sub

Quando você modifica um menu de atalho, essa modificação tem efeito **BRE-SE**

até você reiniciar o Excel. Em outras palavras, menus de atalho
modifi-LEM

cados não se reconfiguram quando você fecha a pasta de trabalho que contém o código VBA. Portanto, se escrever código para modificar um menu de atalho, quase sempre você escreve código para inverter o efeito de sua modificação.

O procedimento DeleteFromShortcut remove o novo item de menu.

Sub DeleteFromShortcut()

On Error Resume Next

Application.CommandBars("Cell").Controls _

("&Change Case").Delete

End Sub

A Figura 19-8 mostra o novo item do menu exibido depois de clicar uma célula com o botão direito.

Figura 19-8: O

menu de

atalho Cell

(Célula)

exibindo um

item do menu

personalizado.

O primeiro comando depois da declaração de um par de variáveis chama o procedimento DeleteFromShortcut. Essa declaração garante que apenas um item Change case aparece no menu de atalho. Tente comentar essa linha (ponha um apóstrofo no início da linha) e rode o

procedimento algumas vezes — agora, não se entusiasme! Clique uma célula com o botão direito e poderá ver múltiplas cópias do item de ExcelVBABokks.indb 327

15/02/2013 14:00:20

328 Parte IV: Como se Comunicar com Seus Usuários menu Change Case. Livre-se de todas as entradas, executando

DeleteFromShortcut várias vezes (uma vez para cada item extra do menu).

Por fim, você precisa de uma maneira para acrescentar o item de menu de atalho quando a pasta de trabalho estiver aberta, e apagar o item de menu quando a pasta de trabalho for fechada. Fazer isso é fácil... se você leu o Capítulo 11. Basta adicionar dois procedimentos de evento ao módulo de código ThisWorkbook: Private Sub Workbook_Open()

Call AddToShortcut

End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)

Call DeleteFromShortcut

End Sub

O procedimento Workbook_Open é executado quando a pasta de trabalho está aberta, e o procedimento Workbook_BeforeClose é executado antes da pasta de trabalho ser fechada. Exatamente o que o médico recomendou.

Desativando um menu de atalho

Se tiver inclinação, você pode desativar um menu de atalho inteiro. Por exemplo, pode fazê-lo para exibir o menu de atalho ao clicar com o botão direito em uma célula. O seguinte procedimento, que é executado automaticamente quando a pasta de trabalho é aberta, desativa o menu de atalho de célula: Private Sub Workbook_Open()

```
Application.CommandBars("Cell").Enabled = False
```

```
End Sub
```

E aqui está o procedimento companheiro dele, que habilita o menu de atalho quando a pasta de trabalho é fechada.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
```

```
Application.CommandBars("Cell").Enabled = True
```

```
End Sub
```

Tenha em mente que mudanças a CommandBars são permanentes. Se **CUIDADO!**

you não executar o procedimento para habilitar o menu de atalho, tal menu de atalho não estará disponível até você reiniciar o Excel. O

procedimento ResetAll, mencionado anteriormente neste capítulo, mostra como você consegue que todos os seus CommandBars voltem às suas posições originais.

ExcelVBABokks.indb 328

15/02/2013 14:00:20



Capítulo 19: Como Acessar suas Macros através da Interface de Usuário 329

Criando uma Barra de Ferramentas

Personalizadas

Se você verificou todos os CommandBars disponíveis, pode ter notado um chamado Built-in Menus (Menus Integrados). Esse CommandBar contém todos os comandos do antigo Excel 2003. A Figura 19-9 mostra parte desse enorme menu pop-up (instantâneo). Todos os antigos comandos estão lá, mas eles não estão muito bem organizados.

Figura 19-9:

Exibindo o

menu de

atalho

Built-in

Menus.

Neste exemplo, apresento um procedimento VBA que (de certa forma) cria a barra de menu do antigo Excel 2003. Ele cria um novo CommandBar e depois, copia os controles Built-in Menus do CommandBar.

Sub MakeOldMenus()

Dim cb As CommandBar

Dim cbc As CommandBarControl

‘ Delete, se existir

On Error Resume Next

Application.CommandBars(“Old Menus”).Delete

On Error GoTo 0

‘ crie uma barra de ferramentas em estilo antigo

‘ Configure a última declaração para Falsa para um menu
mais compacto

Set OldMenu = Application.CommandBars.Add

_(“Old Menus”, , True)

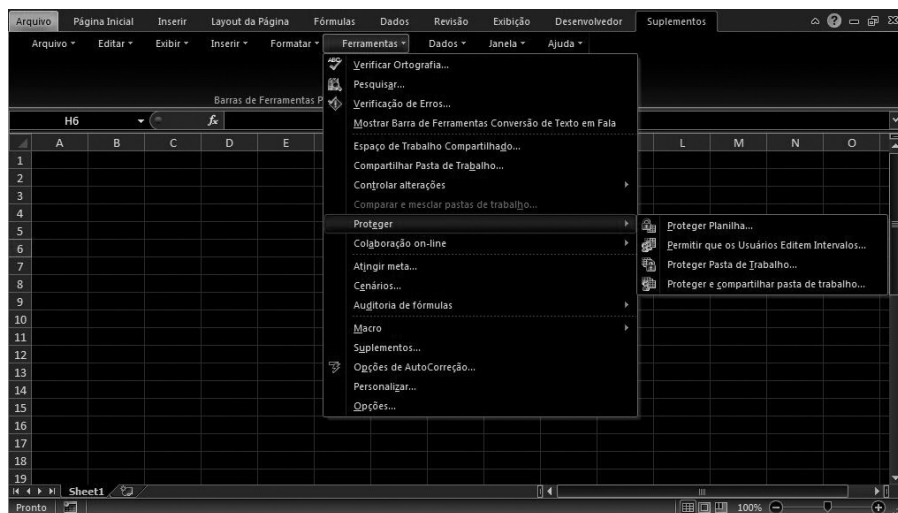
‘ Copie os controles dos menus do Excel”

shortcut menu

With CommandBars(“Built-in Menus”)

ExcelVBABokks.indb 329

15/02/2013 14:00:20



330 Parte IV: Como se Comunicar com Seus Usuários

.Controls(“&Arquivo”).Copy OldMenu

.Controls(“&Editar”).Copy OldMenu

.Controls(“&Exibir”).Copy OldMenu

```
.Controls("&Inserir").Copy OldMenu
```

```
.Controls("&Formatar").Copy OldMenu
```

```
.Controls("&Ferramentas").Copy OldMenu
```

```
.Controls("&Dados").Copy OldMenu
```

```
.Controls("&Janela").Copy OldMenu
```

```
.Controls("&Ajuda").copy OldMenu
```

```
End With
```

```
,
```

Torne visível. Aparece nas abas inseridas

```
Application.CommandBars("Old Menus").Visible = True
```

```
End Sub
```

A Figura 19-10 mostra o resultado de rodar o procedimento

MakeOldMenus. Observe que ele aparece no grupo Custom ToolBars (Barras de Ferramentas Personalizadas) da guia Add-Ins. Afinal, este menu é uma barra de ferramentas. Ele apenas parece um menu.

Figura 19-10:

Uma barra
de ferramentas fingindo
ser o menu
do sistema
do Excel
2003.

O novo CommandBar é chamado de Old Menus. O procedimento começa apagando aquela barra de ferramentas, se ela ainda existir.

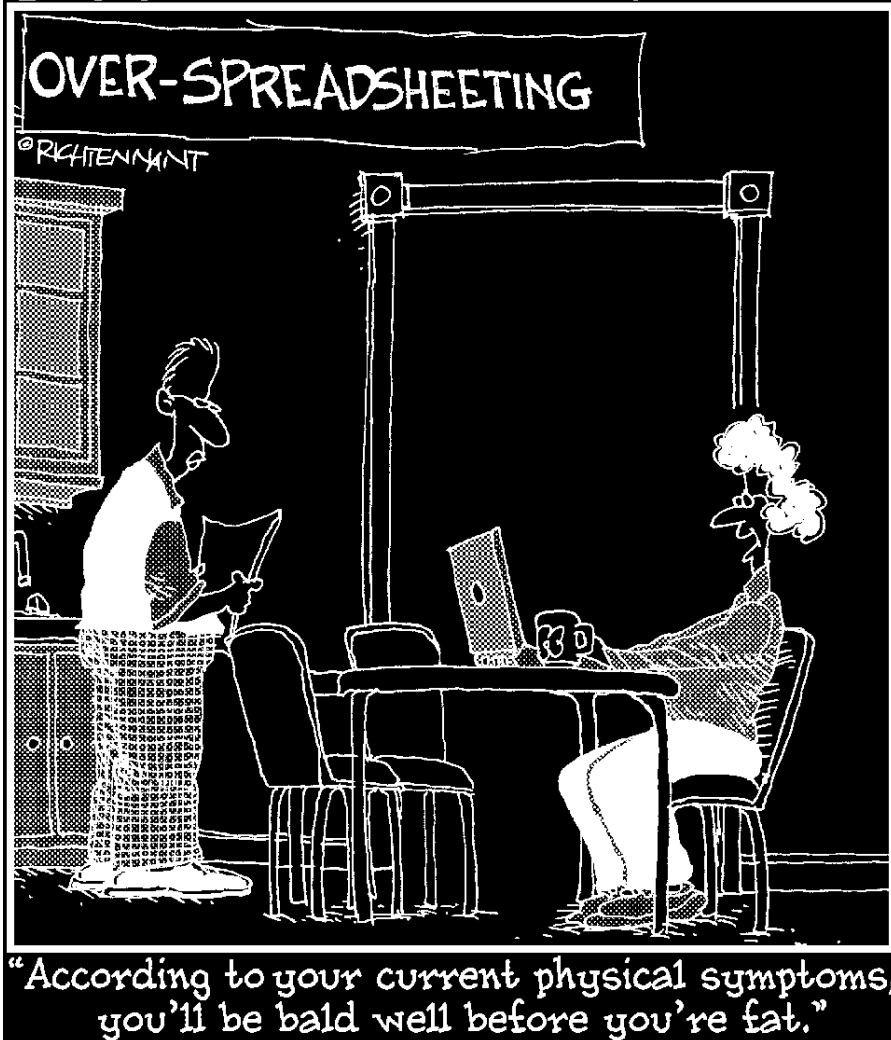
Depois, o código copia os controles de “alto nível” do CommandBar chamado Built-in Menus para Old Menus CommandBar. Por padrão, um novo CommandBar está oculto, assim, a declaração final o torna visível.

O novo “menu” do sistema não é perfeito. Alguns comandos não funcionam. Você perceberá também que os arquivos recentes no menu Arquivo só mostram contentores de lugar. Para se livrar da barra de ferramentas, clique-a com o botão direito e escolha Delete Custom Toolbar.

Este exemplo está disponível no site do livro.

ExcelVBABokks.indb 330

15/02/2013 14:00:20



Parte V

Juntando Tudo

A 5ª Onda

Por Rich Tennant

ExcEsso dE Planilhas

“De acordo com os seus sintomas físicos atuais,
você ficará careca antes de ficar gordo”.

15/02/2013 14:00:21

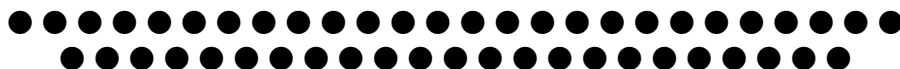
Nesta parte...

O s 19 capítulos anteriores abrangem um bom punhado de material. A essa altura, você ainda pode se sentir um pouco desarticulado sobre toda a coisa de VBA. Os capítulos nesta parte preenchem os espaços e juntam tudo. Você aprende como incluir os seus próprios botões personalizados na interface de usuário do Excel. Eu discuto sobre funções personalizadas de planilha (um recurso muito útil), descrevo add-ins, ofereço mais exemplos de programação e encerro com uma discussão de aplicativos orientados por usuário.

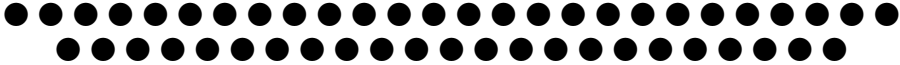
15/02/2013 14:00:21

Capítulo 20

Como Criar Funções de Planilha — e Viver para Contar

*Neste Capítulo*

- ☐ Como saber porque funções personalizadas de planilha são tão úteis
- ☐ Explorando funções que usam vários tipos de argumentos
- ☐ Como entender a caixa de diálogo Inserir Função



Para muitas pessoas, o principal atrativo do VBA é a capacidade de criar funções personalizadas de planilha — funções que parecem, funcionam e se acreditam exatamente como aquelas que a Microsoft integrou ao Excel. Uma *função personalizada* oferece a vantagem adicional de trabalhar exatamente como você quer que ela faça (porque *you* a escreveu). Apresentei funções personalizadas no Capítulo 6. Neste capítulo, eu me aprofundo e descrevo alguns truques do negócio.

Por Que Criar Funções Personalizadas?

Sem dúvida você está familiarizado com as funções de planilha do Excel

— mesmo os iniciantes em Excel sabem como usar funções comuns de planilha, tais como Soma, Média e SE. Pelas minhas contas, o Excel 2007

contém mais de 350 funções de planilha pré-definidas, e o Excel 2010

acrescenta mais umas 50. E se isso não for suficiente, você pode criar funções usando o VBA.

Com todas as funções disponíveis em Excel e VBA, você pode imaginar por que precisaria criar funções. A resposta: para simplificar o seu trabalho. Com um pouco de planejamento, funções personalizadas são muito úteis em fórmulas de planilha e procedimentos VBA. Por exemplo, com frequência, você pode encurtar significativamente uma fórmula, criando uma função personalizada. Afinal, fórmulas mais curtas são mais legíveis e mais fáceis para trabalhar.

ExcelVBABokks.indb 333

15/02/2013 14:00:21

334 Parte V: Juntando Tudo

O que funções personalizadas de planilha podem fazer

Quando você desenvolve funções personalizadas para usar em suas fórmulas de planilha, é importante que você entenda um ponto chave. Os procedimentos Function de planilha VBA são essencialmente passivos. Por exemplo, o código dentro de um procedimento Function

não pode manipular faixas, mudar formatação ou fazer muitas outras coisas que são possíveis com um procedimento Sub. Um exemplo pode ajudar.

Pode ser útil criar uma função que mude a cor de texto em uma célula, com base no valor da célula. No entanto, mesmo tentando, você não pode escrever tal função. Ela sempre retorna um erro de valor.

Lembre-se apenas disto: uma função usada em uma fórmula de planilha retorna um valor — ela não executa ações com objetos.

Dito isto, há algumas exceções a essa regra. Por exemplo, eis um procedimento Function que muda o texto em um comentário de célula:

```
Function ChangeComment(cell, NewText)
```

```
cell.Comment.Text
```

```
NewText
```

```
End Function
```

E eis a fórmula que usa a função. Ela supõe que a célula A1 já tem um comentário. Quando a fórmula é calculada, o comentário é alterado.

```
=ChangeComment (A1, "I changed the comment!")
```

Eu não tenho certeza se isso é um descuido ou um recurso. Mas, é um exemplo raro de uma função VBA que muda alguma coisa em uma planilha.

Como Entender os Princípios Básicos

de Função VBA

Hora de uma rápida revisão. Uma *função* VBA é um procedimento que é armazenado em um módulo VBA. Você pode usar essas funções em

outros procedimentos VBA ou em suas fórmulas de planilha.

Um *módulo* pode conter qualquer quantidade de funções. Você pode usar uma função personalizada em uma fórmula exatamente como se ela fosse uma função integrada. Mas, se a função for definida em uma pasta de trabalho diferente, você deve preceder o nome da função com o nome da pasta de trabalho. Por exemplo, digamos que você

desenvolveu uma função chamada DiscountPrice (que toma um argumento) e a função está armazenada em uma pasta de trabalho chamada pricing.xlsm.

Para usar essa função na pasta de trabalho pricing.xlsm, entre com uma fórmula como esta:

=DiscountPrice(A1)

Se você quiser usar essa função em uma pasta de trabalho *diferente*, entre com uma fórmula tal como esta:

=pricing.xlsm!discountprice(A1)

ExcelVBABokks.indb 334

15/02/2013 14:00:21

Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar 335

Se a função personalizada estiver armazenada em um add-in, você não **DICA**

precisa preceder o nome da função com o nome da pasta de trabalho.

Eu discuto add-ins no Capítulo 21.

As funções personalizadas aparecem na caixa de diálogo Inserir Função, na categoria User Defined (Definido por Usuário). Pressionar Shift + F3 é uma maneira de exibir a caixa de diálogo Inserir Função.

Escrevendo Funções

Lembre-se que o nome de uma função age como uma variável. O valor final dessa variável é o valor retornado pela função. Para demonstrar, examine a seguinte função, a qual retorna o primeiro nome do usuário: Function FirstName()

Dim FullName As String

Dim FirstSpace As Integer


```
FullName = Application.UserName
```

```
FirstSpace = InStr(FullName, " ")
```

```
If FirstSpace = 0 then
```

```
    FirstName = FullName
```

```
Else
```

```
    FirstName = Left(FullName, FirstSpace - 1)
```

```
End If
```

```
End Function
```

Esta função começa designando a propriedade `UserName` do objeto `Application` a uma variável chamada `FullName`. Em seguida, ela usa a função VBA `InStr`, para localizar o primeiro espaço no nome. Se não houver espaço, `FirstSpace` é igual a 0, e `FirstName` é igual ao nome inteiro. Se `FullName` *não* tiver um espaço, a função `Left` extrai o texto para a esquerda do espaço e o atribui a `FirstName`.

Observe que `FirstName` é o nome da função e também é usado como um nome variável *na* função. O valor final de `FirstName` é o valor retornado pela função. Vários cálculos intermediários podem estar acontecendo na função, porém, ela sempre retorna o último valor designado à variável, que é igual ao nome da função.

Todos os exemplos deste capítulo estão disponíveis no site deste livro.

Trabalhando com Argumentos de Função

Para trabalhar com funções, você precisa entender como trabalhar

com argumentos. Os seguintes pontos aplicam-se aos argumentos para as funções de planilha do Excel e funções VBA personalizadas.

ExcelVBABokks.indb 335

15/02/2013 14:00:21

336 Parte V: Juntando Tudo

5 *Argumentos* podem ser referências a células, variáveis (incluindo arrays), constantes, valores literais ou expressões.

5 Algumas funções não têm argumentos.

5 Algumas funções têm um número fixo de argumentos exigidos (de 1 a 60).

5 Algumas funções têm uma combinação de argumentos exigidos e opcionais.

Exemplos de Função

Os exemplos nesta seção demonstram como trabalhar com os vários tipos de argumentos.

Uma função sem argumento

Como procedimentos sub, procedimentos Function não precisam ter argumentos. Por exemplo, o Excel tem algumas funções de planilha integradas que não usam argumentos, incluindo Aleatório, Hoje e Agora.

Eis um exemplo de uma função sem argumentos. A seguinte função retorna a propriedade UserName do objeto Application. O nome do usuário aparece na guia Geral da caixa de diálogo Opções do Excel (no Excel 2007, ela é chamada de guia Popular). Este exemplo simples, mas útil, mostra a única maneira pela qual você pode conseguir que o

nome do usuário apareça em uma célula de planilha: Function User()

‘ Retorna o nome do usuário atual

User = Application.UserName

End Function

Quando você entra com a seguinte fórmula em uma célula de planilha, a célula exibe o nome do usuário atual:

= User()

Tal como acontece com as funções integradas do Excel, você deve incluir parênteses vazios ao usar uma função sem argumentos. Caso contrário, o Excel tenta interpretar a função como uma faixa nomeada.

Uma função com um argumento

A função de argumento único nesta seção destina-se aos gerentes de vendas que precisam calcular as comissões de seus vendedores. A taxa de comissão depende do volume mensal de vendas; aqueles que vendem mais ganham uma taxa mais alta de comissão. A função

ExcelVBABokks.indb 336

15/02/2013 14:00:22

Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar 337

retorna o valor da comissão com base nas vendas mensais (que é o único argumento da função — um argumento exigido). Os cálculos neste exemplo são baseados na Tabela 20-1.

Tabela 20-1 Taxas de Comissão por Vendas

Vendas Mensais

Taxa de Comissão

\$0-\$9.999 8.0%

\$10.000-\$19.999

10.5%

\$20.000-\$39.999 12.0%

\$40.000 +

14.0%

Você pode usar várias abordagens para calcular comissões para os valores de vendas inseridos em uma planilha. Você *poderia* escrever uma longa fórmula de planilha, tal como esta:

=SE(E(A1 >= 0;A1 <= 9999.99);A1*0;08;IF(E(A1 >= 10000;

A1 <= 19999.99);A1*0.105;SE(E(A1 >= 20000;A1 <= 39999.99);
A1*0;12;SE(A1 >= 40000;;A1*9.14;0))))

Várias razões tornam isso uma péssima abordagem. Primeiro, a fórmula é excessivamente complexa. Segundo, os valores são codificados com dificuldade na fórmula, tornando-a difícil de modificar caso a estrutura de comissão mude.

Uma abordagem melhor é criar uma tabela de valores de comissão e usar a função de planilha PROCV para calcular as comissões:

=PROCV(A;Table;2)*A1

Uma outra abordagem, que não requer uma tabela de comissões, é criar uma função personalizada:

Function Commissuin(Sales)

‘ Calcula as comission de vendas

Const Tier1 As Double = 0.08

Const Tier1 As Double = 0.105

Const Tier3 As Double = 0.12

Const Tier4 As Double = 0.14

Select Case Sales

Case 0 To 999.99: Commission = Sales * Tier1

Case 10000 To 19999.00: Commission = Sales * Tier2

Case 20000 To 39999.99: Commission = Sales * Tier3

Case Is >= 40000: Commission = Sales * Tier4

End Select

Commission = Round(Commission, 2)

End Function

ExcelVBABokks.indb 337

15/02/2013 14:00:22

	A	B	C	D
1	Name	Sales	Commission	
2	Adams	\$61.983,00	\$8.677,62	
3	Baker	\$3.506,00	\$280,48	
4	Douglas	\$38.973,00	\$4.676,76	
5	Emmett	\$32.092,00	\$3.851,04	
6	Franklin	\$27.354,00	\$3.282,48	
7	Johnson	\$17.833,00	\$1.872,46	
8	Kent	\$41.598,00	\$5.823,72	
9	Mays	\$32.000,00	\$3.840,00	
10	Quincy	\$5.000,00	\$400,00	
11	Randall	\$68.793,00	\$9.631,02	
12	Smith	\$31.093,00	\$3.731,16	
13	Walker	\$24.509,00	\$2.941,08	
14	Zeller	\$41.544,00	\$5.816,16	
15				

338 Parte V: Juntando Tudo

Observe que as quatro taxas de comissão são declaradas como constantes, ao invés de código difícil. Isso facilita muito modificar a função se as taxas de comissão mudarem.

Depois de definir essa função em um módulo VBA, você pode usá-la em uma fórmula de planilha. Entrar com a fórmula a seguir em uma célula produz um resultado de 3.000. A quantia de 25000 qualifica para uma taxa de comissão de 12 por cento: =Commission(25000)

A Figura 20-1 mostra uma planilha que usa a função Commission em fórmulas na coluna C.

Figura 20-1:

Usando a

função

Commission

em uma

planilha.

Uma função com dois argumentos

O seguinte exemplo se baseia no anterior. Imagine que o gerente de vendas implemente uma nova política para recompensar funcionários antigos: o total da comissão paga aumenta em 1% por cada ano que o vendedor está na empresa.

Eu modifiquei a função personalizada *Commission* (definida na seção anterior) para que ela tome dois argumentos, sendo que ambos são argumentos exigidos. Chame esta nova função de *Commission2*:
Function *Commission2*(Sales, Years) ‘ Calcula a comissão de vendas baseada nos anos de

serviço

Const Tier1 As Double = 0.08

Const Tier2 As Double = 0.105

Const Tier3 As Double = 0.12

Const Tier4 As Double = 0.14

ExcelVBABokks.indb 338

15/02/2013 14:00:22

Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar 339

Select Case Sales

Case 0 to 9999.99: $Commission2 = Sales * Tier1$

Case 10000 To 19999.99: $\text{Commission2} = \text{Sales} * \text{Tier2}$

Case 20000 to 39999.99: $\text{Commission2} = \text{Sales} * \text{Tier3}$

Case Is > = 40000: $\text{Commission2} = \text{Sales} * \text{Tier4}$

End Select

$\text{Commission2} = \text{Commission2} + (\text{Commission2} * \text{Years} / 100)$

$\text{Commission2} = \text{Round}(\text{Commission2}, 2)$

End Function

Eu apenas adicionei o segundo argumento (Years) à declaração Function (Função) e incluí um cálculo adicional, que ajusta a comissão antes de encerrar a função. Esse cálculo multiplica a comissão original pelo número de anos de serviços, divide por 100 e, depois, acrescenta o resultado ao cálculo original.

Eis um exemplo de como você pode escrever uma fórmula usando esta função (ela supõe que a quantia de vendas está na célula A1; a célula B1

especifica o número de anos que o vendedor trabalhou).

$= \text{Commission2}(\text{A1}; \text{B1})$

Uma função com um argumento faixa

Usar uma faixa de planilha como um argumento não é tão complicado; o Excel cuida dos detalhes por trás.

Suponha que você queira calcular a média dos cinco maiores valores em uma faixa chamada Data (Dados). O Excel não tem uma função que possa fazer isso, assim, provavelmente você escreveria uma

fórmula:

= (MÁXIMO(Data;1) + MÁXIMO(Data;2) + MÁXIMO(Data;3) +

MÁXIMO(Data;4) + MÁXIMO(Data;5))/5

Esta fórmula usa a função Máximo do Excel, a qual retorna o valor mais alto em uma faixa. A fórmula adiciona os cinco maiores valores na faixa chamada Data e, depois, divide o resultado por 5. A fórmula funciona bem, mas é bem pesada. E se você decidir que precisa computar a média dos seis maiores valores? Precisaria reescrever a fórmula — e garantir que atualizaria todas as cópias da fórmula.

Não seria mais fácil se o Excel tivesse uma função chamada TopAvg (média mais alta)? Assim, você computaria a média usando a seguinte (inexistente) função:

=TopAvg(Data;5)

ExcelVBABokks.indb 339

15/02/2013 14:00:22

340 Parte V: Juntando Tudo

Este exemplo mostra uma situação em que uma função personalizada pode tornar as coisas muito mais fáceis para você. A função VBA a seguir, chamada TopAvg, retorna a média dos maiores valores *N* em uma faixa: Function TopAvg(InRange, N)

‘ Retorna a média dos maiores valores de N em

InRange

Dim Sum As Double

Dim I As Long

Sum = 0

For i = 1 To N

Sum = Sum + WorksheetFunction.Large(InRange, i)

Next i

TopAvg = Sum / N

End Function

Esta função toma dois argumentos: InRange (que é uma faixa de planilha) e N (o número de valores para tirar a média). Ela começa inicializando a variável Sum para 0. Depois, usa um loop For-Next para calcular a soma dos maiores valores *N* na faixa. Observe que eu uso a função Máximo (Large) do Excel dentro do loop. Por fim, TopAvg é atribuída ao valor de Sum dividido por N.

Você pode usar todas as funções de planilha do Excel em seus procedimentos VBA, *exceto* aqueles que têm equivalentes em VBA. Por exemplo, o VBA tem uma função Rnd, que retorna um número aleatório.

Portanto, você não pode usar a função Aleatório (rand) do Excel em um procedimento VBA.

Uma função com um argumento opcional

Muitas funções de planilha integradas do Excel usam argumentos opcionais. Um exemplo é a função LEFT (Esquerda), a qual retorna caracteres do lado esquerdo de uma string. Em seguida, a sua sintaxe oficial: ESQUERDA(texto,[núm_corect]) O primeiro argumento é exigido, mas o segundo (entre colchetes) é opcional. Se você omitir o argumento opcional, o Excel assume o valor de 1. Portanto, as seguintes fórmulas retornam o mesmo resultado: =ESQUERDA(A1;1)

=ESQUERDA(A1)

As funções personalizadas que você desenvolve em VBA também podem ter argumentos opcionais. Você especifica um argumento

ExcelVBABokks.indb 340

15/02/2013 14:00:23

Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar 341

opcional, precedendo o nome do argumento com a palavra-chave

Optional, seguido por um sinal de igual e o valor padrão. Se o argumento opcional estiver faltando, o código usa o valor padrão.

O seguinte exemplo mostra uma função personalizada que usa um argumento opcional:

```
Function DrawOne(InRange, Optional Recalc = 0)
```

```
‘ Escolhe uma célula aleatoriamente de uma faixa
```

```
Randomize
```

```
‘ Torna a função volátil se Recalc é igual a 1
```

```
If Recalc = 1 Then Application.Volatile True
```

```
‘ Determine a random cell
```

```
DrawOne = InRange(Int((InRange.Count) * Rnd + 1))
```

```
End Function
```

Esta função escolhe aleatoriamente uma célula a partir de uma faixa de entrada. A faixa passada como um argumento é, na verdade, um array (eu explico arrays no Capítulo 7), e a função seleciona, aleatoriamente, um item do array. Se o segundo argumento for 1, o valor selecionado muda sempre que a planilha for recalculada (a função se torna *volátil*.) Se o segundo argumento for 0 (ou for

omitido), a função não é recalculada, a menos que uma das células na faixa de entrada seja modificada.

Como depurar funções personalizadas

Depurar um procedimento Function pode ser um pouco mais desafiador do que depurar um procedimento Sub. Se você desativar um procedimento Sub, se você desativar uma função para usar em fórmulas que for avaliada — o que poderia ser bem desagradável.

Teste o procedimento, chamando-o a partir de um procedimento Sub. Erros

(normalmente #VALUE!). Em outras palavras, você não recebe uma mensagem normalmente em uma janela instantânea

mal de erro em tempo de execução que o

(pop-up), e você pode corrigir o problema (ajuda a localizar a declaração afetada).

ma (se souber qual é) ou pular direto

Você pode escolher entre três métodos para

para o depurador.

depurar funções personalizadas:

5 Configure um ponto de interrupção na

5 Coloque as funções MsgBox em luga—

função e depois, use o depurador do

res estratégicos para monitorar o valor

Excel para percorrer a função. Depois,

de variáveis específicas. Felizmente, as

você pode acessar todas as ferramen—

caixas de mensagem nos procedimen—

tas habituais do depurador. Consulte o

tos Function aparecem quando você

Capítulo 13 para mais detalhes sobre o

executa o procedimento. Assegure-se

depurador.

ExcelVBABokks.indb 341

15/02/2013 14:00:23

342 Parte V: Juntando Tudo

Eu uso a declaração Randomize para garantir que um número aleatório diferente “pré-selecionado” seja escolhido cada vez que a pasta de trabalho for aberta. Sem essa declaração, os mesmos números aleatórios serão gerados sempre que a pasta de trabalho for aberta.

Você pode usar essa função para escolher números da loteria, selecionando um vencedor a partir da lista de nomes, e assim por diante.

Uma função com um número

indefinido de argumentos

Algumas funções de planilha do Excel tomam um número indefinido

de argumentos. Um exemplo conhecido é a função SOMA, que tem a seguinte sintaxe:

SOMA(núm1,núm2...)

O primeiro argumento é exigido, mas você pode ter até 254 argumentos adicionais. Eis um exemplo de uma função SOMA com quatro faixas de argumentos:

=SOMA(A1:A5;C1:C5;E1:E5;G1:G5)

Eis uma função VBA que pode ter qualquer quantidade de argumentos de um único valor. Esta função não funciona com argumentos de faixa de múltiplas células.

Function Concat(string1, ParamArray string2())

‘ Demonstra um número indefinido de declarações de funções

Dim Args As Variant

‘ Processa as primeiras declarações

Concat = string1

‘ Process additional arguments (if any)

If UBound(string2) < > -1 Then

For Args = LBound(string2) To Ubound(string2)

Concat = Concat & “ “ & string2(Args)

Next Args

End If

End Function

Esta função é semelhante à função CONCATENAR do Excel, a qual combina argumentos de texto em uma única string. A diferença é que essa função personalizada insere um espaço entre cada par de strings concatenadas.

O segundo argumento, string2(), é um array precedido pela palavra-chave ParamArray. Se o segundo argumento estiver vazio, a função UBound retorna -1 e a função encerra. Se o segundo argumento não

ExcelVBABokks.indb 342

	A	B	C	D
1	Title	First	Last	Concat
2	Mr.	Jim	Smith	Mr. Jim Smith
3	Dr.	Tina	Peterson	Dr. Tina Peterson
4	Ms.	Jane	Doe	Ms. Jane Doe
5		Frank	Franklin	Frank Franklin
6	Mr.	Willie	Nielson	Mr. Willie Nielson
7	Mrs.	Steve	Marks	Mrs. Steve Marks
8		Hank	Walker	Hank Walker
9	Mr.	Bill		Mr. Bill
10				
11				

Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar 343

estiver vazio, o procedimento faz loop através dos elementos do array string2 e processa cada argumento adicional. As funções LBound e UBound determinam o início e o fim dos elementos do array. Normalmente, o elemento de início é 0, a menos que você o declare

como alguma outra coisa ou que use uma declaração Option Base 1 no início do seu módulo.

ParamArray só pode ser aplicado ao *último* argumento no procedimento.

BRE-SE

Ele é sempre um tipo de dados Variant e é sempre um argumento opcional **LEM**

(ainda que você não use a palavra-chave Optional). A Figura 20-2 mostra essa função em uso. Examine a figura para ver os resultados diferentes daqueles produzidos pela função CONCATENAR do Excel, que não inserem um espaço entre os itens concatenados.

Figura 20-2:

Usando a

função

Concat em

fórmulas

de planilha.

Funções Que Retornam um Array

As fórmulas de array (arr]anjo: coleção de dados similares armazenados sob o mesmo nome) são um dos recursos mais poderosos do Excel.

Se você está familiarizado com fórmulas de array, ficará feliz em saber que pode criar funções VBA que retornam um array.

Retornando um array de nomes de meses

Começarei com um simples exemplo. A função MonthNames retorna um array de 12 elementos de — você adivinhou — nomes de meses.

Function MonthNames()

MonthNames = Array(“Janeiro”, “Fevereiro”, “Março”, _

“Abril”, “Maio”, “Junho”, “Julho”, “Agosto”, _

“Setembro”, “Outubro”, “Novembro”, “Dezembro”)

End Function

Para usar a função MonthNames em uma planilha, você deve entrar com ela como uma fórmula de array de 12 células. Por exemplo, selecione a faixa A1:L1 e entre com **=MonthNames()**. Depois, use Ctrl + Shift + Enter para entrar com a fórmula de array em todas as 12 células selecionadas. A Figura 20-3 mostra o resultado.

ExcelVBABokks.indb 343

15/02/2013 14:00:23

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Janeiro	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro	
2													
3													
4													

344 Parte V: Juntando Tudo

Figura 20-3:

Usando a

função

MonthNames para

retornar um

array de 12

elementos.

Se você quiser exibir os nomes de meses em uma coluna, selecione 12 células em uma coluna e use esta fórmula de array (não se esqueça de entrar com ela usando Ctrl + Shift + Enter):

=TRANSPOR(MonthNames())

Retornando uma lista classificada

Suponha que você tem uma lista de nomes que deseja mostrar em ordem classificada em uma outra faixa de células. Não seria ótimo ter uma função de planilha para fazer isso por você?

A função personalizada nesta seção faz exatamente isso: ela toma uma faixa de células de coluna única como seu argumento e, depois, retorna um array daquelas células classificadas. A Figura 20-4 mostra como funciona. A faixa A2:A13 contém alguns nomes. A faixa C2:C13 contém essa fórmula de array de múltiplas células (lembre-se, a fórmula deve ser inserida pressionando Ctrl + Shift + Enter).

=Sorted(A2:A13)

Eis o código para a função Sorted:

Function Sorted(Rng As Range)

Dim SortedData() As Variant

Dim Cell As Range

Dim Temp As Variant, i As Long, j As Long

Dim NonEmpty As Long

‘ transfere os dados para SortedData

For Each Cell In Rng

If Not IsEmpty(Cell) Then

NonEmpty = NonEmpty + 1

ReDim Preserve SortedData(1 To NonEmpty)

SortedData(NonEmpty) = Cell.Value

End If

Next Cell

ExcelVBABokks.indb 344

15/02/2013 14:00:24

	A	B	C	D
1	Não classificado		Classificado	
2	Keith		Abigail	
3	Frank		Ann	
4	Jackie		Darren	
5	Tim		Frank	
6	Ann		Jackie	
7	Louise		Keith	
8	Zola		Louise	
9	Opie		Mary	
10	Ralph		Opie	
11	Mary		Ralph	
12	Abigail		Tim	
13	Darren		Zola	
14				

Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar 345

‘ Organiza o array

For i = 1 To NonEmpty

For j = i + 1 To NonEmpty

If SortedData(i) > SortedData(j) Then

Temp = SortedData(j)

SortedData(i) = Temp

End If

Next j

Next i

‘ Transporta o array e

Sorted = Application.Transpose(SortedData)

End Function

Figura 20-4:

Usando uma

função

personalizada

para retornar

uma faixa

classificada.

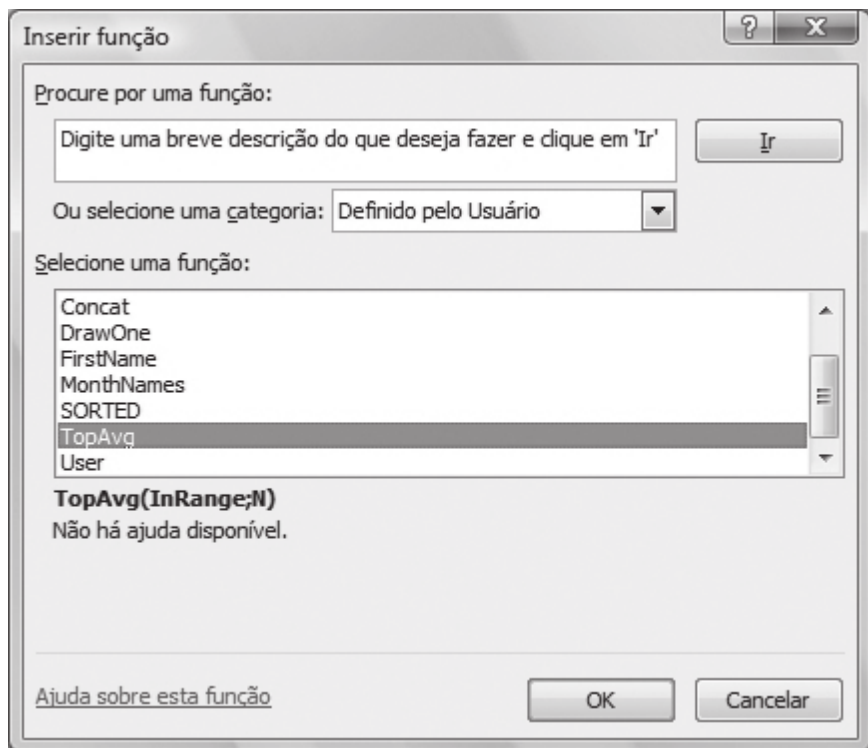
A função Sorted começa criando um array chamado SortedData. Esse array contém todos os valores que não estão em branco na faixa de argumento. Em seguida, o array SortedData é classificado, usando um algoritmo do tipo bolha. Porque o array é um array horizontal, ele deve ser invertido antes de ser retornado pela função.

A função Sorted trabalha com uma faixa de qualquer tamanho, desde que ela esteja em uma coluna ou linha única. Se os dados não classificados estiverem em uma linha, a sua fórmula precisa usar a função TRANSPOR do Excel para exibir horizontalmente os dados classificados. Por exemplo: =TRANSPOR(Sorted(A16:L:16))

Como Usar a Caixa de Diálogo

Inserir Função

A caixa de diálogo Inserir Função do Excel é uma ferramenta útil que permite escolher uma função de planilha a partir de uma lista e solicita que você entre com os argumentos da função. E, como observei anteriormente neste capítulo, as suas funções personalizadas de planilha também aparecem ExcelVBABokks.indb 345



346 Parte V: Juntando Tudo

nessa caixa de diálogo. Funções personalizadas aparecem na categoria Definido pelo Usuário.

Os procedimentos Function definidos com a palavra-chave Private não **DICA**

aparecem na caixa de diálogo Inserir Função. Portanto, se você escrever um procedimento Function que se destine a ser usado apenas por outros procedimentos VBA (mas não em fórmulas), você deve declará-

-lo como Private.

Exibindo a descrição da função

A caixa de diálogo Inserir Função exibe uma descrição de cada função integrada. Mas, como você pode ver na Figura 20-5, uma função personalizada exibe o seguinte texto como sua descrição: não há ajuda disponível.

Figura 20-5:

Por padrão,

a caixa de

diálogo

Inserir

Função não

oferece

uma descri-

ção para

funções

personalizadas.

Para exibir uma descrição significativa de sua função personalizada na caixa de diálogo Inserir Função, execute algumas etapas adicionais (não intuitivas):

1. Ative uma planilha na pasta de trabalho que contém a função personalizada.

2. Escolha Desenvolvedor⇒Código Macros (ou pressione Alt + F8).

A caixa de diálogo Macro aparece.

3. No campo Nome de macro, digite o nome da função.

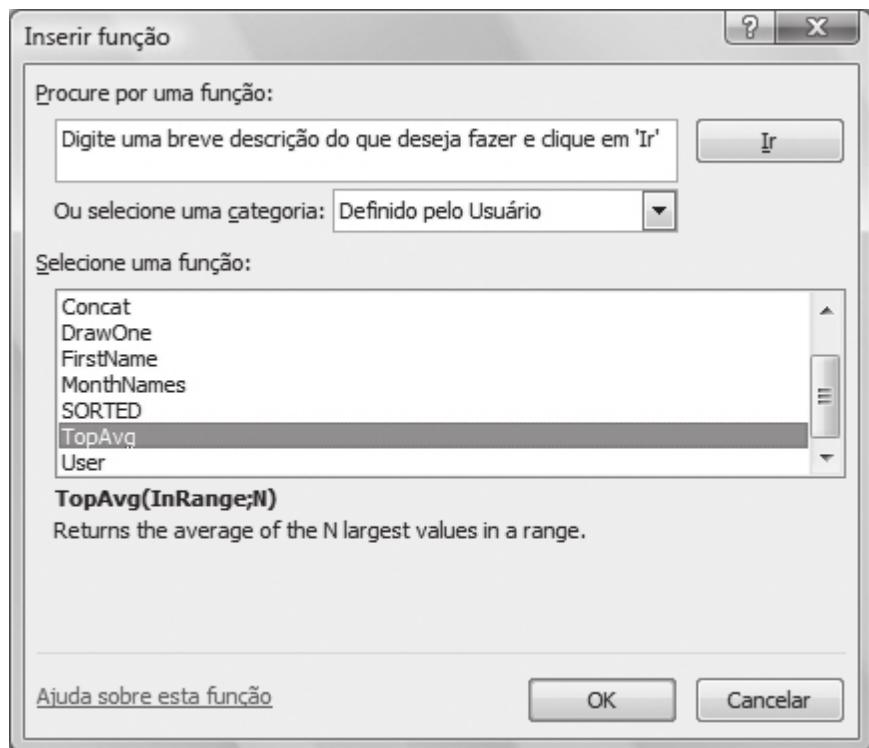
Observe que a função não aparece na lista de macros; você deve digitar o nome.

4. Clique o botão Opções.

A caixa de diálogo Opções de macro aparece.

ExcelVBABokks.indb 346

15/02/2013 14:00:24



Capítulo 20: Como Criar Funções de Planilha — e Viver para Contar 347

5. No campo Descrição, digite uma descrição da função.

6. Clique OK.

7. Clique Cancelar.

Agora, a caixa de diálogo Inserir Função exibe a descrição de sua função; veja a Figura 20-6.

Figura 20-6:

A função

personalizada agora

exibe uma

descrição.

Por padrão, funções personalizadas são listadas na categoria Definido
DICA

pelo Usuário. Para adicionar uma função a uma categoria diferente, você precisa usar o VBA. Esta declaração, quando executada, adiciona a função TopAvg à categoria Matemática e Trigonométrica (que é a categoria #3): Application.MacroOptions Macro: = "TopAvg", Category: = 3

Consulte o sistema de ajuda quanto aos números das outras categoria.

E lembre-se, você só precisa executar essa declaração uma vez. Depois de executá-la (e salvar a pasta de trabalho), o número da categoria é permanentemente designado à função.

Descrições de argumento

Quando você acessar uma função integrada a partir da caixa de diálogo Inserir Função, a caixa de diálogo Argumentos da Função exibe uma descrição de cada argumento (veja a Figura 20-7). Se você usa o Excel 2007, não há uma forma direta de oferecer tais descrições às suas funções personalizadas.

Mas, se você usa o Excel 2010, um novo recurso permite que você especifique descrições para as suas funções personalizadas. Isso é feito usando o método MacroOptions. Eis um exemplo que adiciona descrições aos argumentos usados pela função TopAvg:

ExcelVBABokks.indb 347

15/02/2013 14:00:25

Argumentos da função

PGTO

Taxa	,065/12	= 0,005416667
Nper	360	= 360
Vp		= número
Vf		= número
Tipo		= número

=

Calcula o pagamento de um empréstimo com base em pagamentos e em uma taxa de juros constantes.

Nper é o número total de pagamentos em um empréstimo.

Resultado da fórmula =

[Ajuda sobre esta função](#)

OK Cancelar

348 Parte V: Juntando Tudo

Sub AddArgumentDescriptions()

Application.MacroOptions Macro: = "TopAvg", _

ArgumentDescriptions: = _

Array("Faixa que contém os valores", _

"Número de valores na média")

End Sub

Você só precisa executar este procedimento uma vez. Depois de executá-lo, as descrições do argumento são armazenadas na pasta de trabalho e são associadas à função.

Note que a descrição do argumento aparece como um argumento para a função Array. Você deve usar a função Array mesmo se estiver inserindo uma descrição para uma função que tenha apenas um

argumento.

Figura 20-7:

Por padrão, a

caixa de

diálogo

Argumentos

da função

exibe

descrições

de argumento de função

apenas para

funções

integradas.

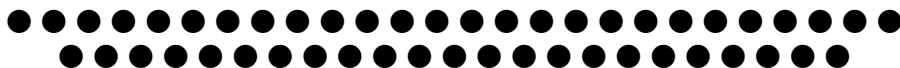
Este capítulo oferece muitas informações sobre a criação de funções personalizadas de planilha. Use esses exemplos como modelos quando criar funções para o seu próprio trabalho. Como sempre, a ajuda online oferece detalhes adicionais. Vá para o Capítulo 21 se quiser descobrir como tornar as suas funções personalizadas mais acessíveis, armazenando-as em um add-in (suplemento).

ExcelVBABokks.indb 348

15/02/2013 14:00:25

Capítulo 21

Criando Add-Ins do Excel

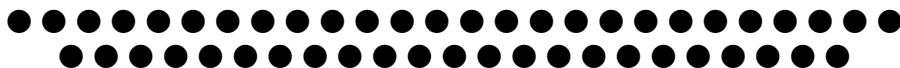


Neste Capítulo

☐ Usando add-ins (suplementos): que conceito!

❑ Saiba por que criar os seus próprios add-ins

❑ Criando add-ins personalizados



Um dos recursos mais engenhosos do Excel — pelo menos na *minha* mente — é a capacidade de criar add-ins (suplementos).

Neste capítulo, explico porque eu gosto desse recurso, e mostro como criar add-ins usando apenas as ferramentas integradas no Excel.

Certo ... Então, o Que é um Add-In?

O que é um add-in? Que bom que você perguntou. Um *add-in* do Excel é algo que você acrescenta para aperfeiçoar a funcionalidade do Excel.

Alguns add-ins oferecem novas funções de planilha que você pode usar em fórmulas; outros add-ins fornecem novos comandos ou utilitários. Se o add-in for adequadamente projetado, os novos recursos se misturam bem com a interface original, para que eles pareçam fazer parte do programa.

O Excel vem com vários add-ins, incluindo o Pacote de Ferramentas de **DICA**

Análise e o Solver. Você também pode conseguir add-ins do Excel de fornecedores terceirizados ou como shareware (programas que podem ser usados e copiados livremente). O meu Power Utility Pak é um exemplo de add-in (ele acrescenta cerca de 70 novos recursos ao Excel, mais um punhado de novas funções de planilha).

Qualquer usuário com conhecimento pode criar add-ins, mas são exigidas habilidades de programação em VBA. Um add-in do Excel é, basicamente, uma forma diferente de arquivo de pasta de trabalho XLSM. Mais especificamente, um add-in é uma pasta de trabalho XLSM

normal, com as seguintes diferenças:

5 A propriedade IsAddin do objeto Workbook é True.

5 A janela de pasta de trabalho é oculta e não pode ser exibida usando o comando Exibição>>Janela>>Reexibir janela.

5 A pasta de trabalho não é um membro da coleção Workbooks.

Ao invés disso, ela está na coleção de AddIns.

ExcelVBABokks.indb 349

15/02/2013 14:00:25

350 Parte V: Juntando Tudo

Você pode converter qualquer arquivo de pasta de trabalho em um add-in, mas nem todas as pastas de trabalho são boas candidatas.

Como os acessórios estão sempre ocultos, você não pode exibir planilhas ou planilhas de gráficos contidas em um add-in. No entanto, você pode acessar os procedimentos Sub e Function VBA de um add-in e exibir caixas de diálogo contidas em UserForms.

Normalmente, os add-ins do Excel têm uma extensão de arquivo XLAM

BRE-SE

para distingui-los dos arquivos de planilha XLSM. Versões anteriores de LEM

Excel criaram add-ins com uma extensão XLA.

Por Que Criar Add-Ins?

Você poderia resolver converter o seu aplicativo em Excel em um add-in por qualquer uma das seguintes razões:

5 Dificultar o acesso ao seu código: Quando você distribui um aplicativo como um add-in (e protege o seu projeto VBA),

usuários casuais não podem ver as planilhas na pasta de

trabalho. Se você usar técnicas proprietárias em seu código

VBA, você pode dificultar que outros copiem o código. Os

recursos de proteção do Excel não são perfeitos e utilitários de invasão de senha estão disponíveis.

5 Evitar confusão: Se um usuário carregar o seu aplicativo como um add-in, o arquivo fica invisível e, portanto, menos passível de confundir usuários iniciantes ou atrapalhar. Diferente de uma pasta de trabalho oculta, o conteúdo de um add-in não pode ser revelado.

5 Simplificar o acesso a funções de planilha: As funções personalizadas de planilha que você armazena em um add-in não requerem o nome qualificador da pasta de trabalho. Por exemplo, se você armazenar uma função personalizada chamada MOVAVG

em uma pasta de trabalho chamada NEWFUNC.XLSM, você deve usar sintaxe como a seguinte para usar esta função em uma diferente pasta de trabalho:

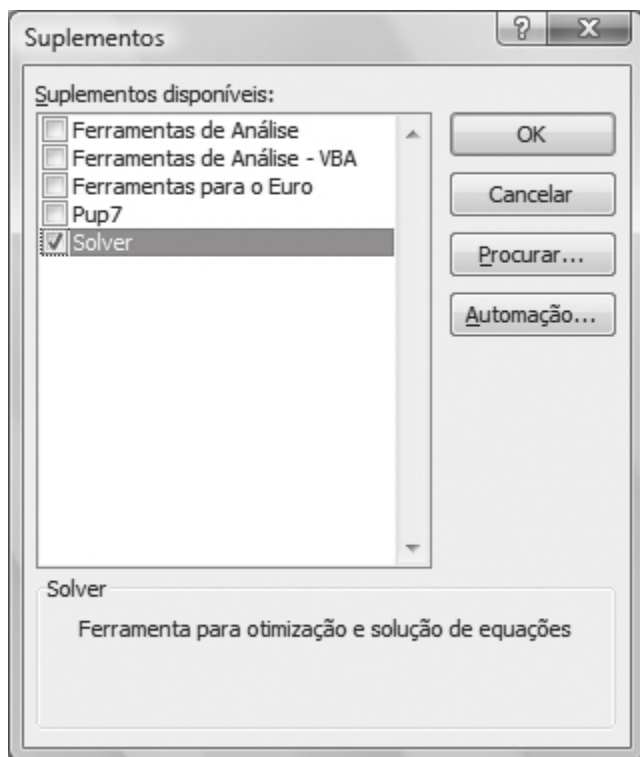
=NEWFUNC.XLSM!MOVAVG(A1:A50)

Mas se esta função for armazenada em um arquivo add-in que está aberto, você pode usar sintaxe muito mais simples, pois não precisa incluir referência ao arquivo:

=MOVAVG(A1:A50)

5 Oferecer acesso mais fácil aos usuários: Depois de identificar a localização do seu add-in, ele aparece na caixa de diálogo Add-Ins, com um nome amigável e uma descrição do que ele faz.

Facilmente, o usuário pode ativar ou desativar o seu add-in.



Capítulo 21: Criando Add-Ins do Excel

351

5 Obter mais controle no carregamento: Os add-ins podem ser abertos automaticamente quando o Excel iniciar, independente do diretório em que eles estão armazenados.

5 Evitar exibir solicitações ao descarregar: Quando um add-in é fechado, o usuário nunca vê a caixa de diálogo aparecer pedindo que você salve as mudanças no arquivo.

Trabalhando com Add-Ins

Você carrega e descarrega add-ins usando a caixa de diálogo Suplementos. Para exibir essa caixa de diálogo, escolha Arquivo ⇨ Opções ⇨ Suplementos. Depois, selecione Suplementos do Excel da lista drop-down dessa caixa de diálogo e clique Ir. Se você estiver usando Excel 2010, chegar a essa caixa de diálogo é um pouco mais fácil: escolha Desenvolve dor⇨Suplementos⇨Suplementos. Porém, o método mais fácil é simplesmente pressionar Alt + MU (o antigo atalho de teclado do Excel 2003).

Qualquer desses métodos exibe a caixa de diálogo Suplementos, mostrada na Figura 21-1. A caixa de listagem contém os nomes de todos os suplementos que o Excel conhece. Nessa lista, marcas de verificação identificam quaisquer add-ins abertos no momento. Você pode abrir e fechar add-ins a partir da caixa de diálogo, selecionando ou desfazendo a seleção das caixas de verificação.

Figura 21-1:

A caixa de
diálogo
Suplementos
(add-ins)
relaciona
todos os
add-ins
conhecidos
pelo Excel.

Você também pode abrir a maioria dos arquivos de add-in (como se **CUIDADO!**

eles fossem arquivos de pasta de trabalho), escolhendo o comando Arquivo⇨Abrir. Um acessório aberto dessa maneira não aparece na caixa de diálogo Suplementos. Além disso, se o add-in foi aberto usando o comando Abrir, você não pode fechá-lo escolhendo Arquivo⇨Fechar.

Você só pode remover o add-in saindo e reiniciando o Excel ou escrevendo uma macro para fechá-lo.

15/02/2013 14:00:26

352 Parte V: Juntando Tudo

Quando você abre um acessório, pode ou não notar alguma coisa diferente. No entanto, em muitos casos, A Faixa de Opções muda de alguma maneira — o Excel exibe ou uma nova guia ou um ou mais novos grupos em uma guia existente. Por exemplo, abrir o add-in Ferramentas de Análise dá a você um novo item na guia Dados: Análise de Dados. Se o add-in só contiver funções de planilha, as novas funções aparecem na caixa de diálogo Inserir Função, e você não verá alteração na interface de usuário do Excel.

Princípios Básicos do Add-In

Embora você possa converter qualquer pasta de trabalho a um add-in, nem todas as pastas de trabalho se beneficiam dessa conversão. Uma pasta de trabalho sem macros torna um acessório completamente inútil.

Na verdade, os únicos tipos de pastas de trabalho que se beneficiam de ser convertidos a um suplemento são os que contêm macros. Por exemplo, uma pasta de trabalho que consiste de macros de objetivo geral (procedimentos Sub e Function) fazem um add-in ideal.

Criar um add-in não é difícil, mas exige um pouco de trabalho extra. Use as seguintes etapas para criar um add-in a partir de um arquivo normal de pasta de trabalho:

1. Desenvolva o seu aplicativo e assegure-se de que tudo funciona adequadamente.

Não se esqueça de incluir um método para executar a macro ou

macros. Você pode definir uma tecla de atalho ou personalizar, de alguma maneira, a interface de usuário (veja o Capítulo 19). Se o add-in consistir apenas de funções, não há necessidade de incluir um método para executá-las, pois elas aparecerão na caixa de diálogo Inserir Função.

2. Teste o aplicativo, executando-o quando uma pasta de trabalho diferente estiver ativa.

Fazer isso simula o comportamento do aplicativo quando ele é usado

como um add-in, pois um add-in nunca é a pasta de trabalho ativa.

3. Ative o VBE e selecione a pasta de trabalho na janela de Projeto; escolha Ferramentas→Propriedade de VBA Project e clique a guia Proteção; selecione a caixa de verificação Bloquear Projeto Para Exibição e entre com uma senha (duas vezes); clique OK.

Esta etapa só é necessária se você quiser evitar que outros vejam ou modifiquem as suas macros ou UserForms.

4. No Excel 2010, escolha Desenvolvedor→Painel de Documentos.

No Excel 2007, escolha Office→Preparar→Propriedades.

O Excel exibe o seu painel Propriedades de Documentos abaixo da Faixa de Opções .

ExcelVBABokks.indb 352

15/02/2013 14:00:26

Capítulo 21: Criando Add-Ins do Excel

353

5. No painel Document Properties, entre com um rápido título descritivo no campo Título e uma descrição mais longa no campo Comentários.

As etapas 4 e 5 não são exigidas, mas tornam o add-in mais fácil de usar, porque as descrições fornecidas aparecem na caixa de

diálogo Add-Ins quando o seu add-in (acessório) é selecionado.

6. Escolha Arquivo→Salvar Como.

7. Na caixa de diálogo Salvar Como, selecione a opção Excel (*.xlam) a partir da lista drop-down Tipo.

8. Especifique a pasta que armazenará o Suplemento.

O Excel propõe uma pasta chamada Suplementos, mas você pode

salvar o arquivo em qualquer pasta que quiser.

9. Clique Salvar.

Uma cópia da sua pasta de trabalho é convertida a um add-in e é salva com a extensão XLAM. A sua pasta de trabalho original permanece aberta.

Um Exemplo de Add-In

Nesta seção, eu discuto as etapas básicas envolvidas na criação de um acessório útil. O exemplo é baseado no utilitário de conversão de texto Change Case, que descrevi no Capítulo 16.

A versão XLSM deste exemplo está disponível no site deste livro. Você pode criar um acessório a partir dessa pasta de trabalho.

Configurando a pasta de trabalho

A pasta de trabalho consiste de uma planilha em branco, um módulo VBA e um UserForm. No Capítulo 19, eu já acrescentei código à pasta de trabalho que cria um novo item no menu de atalho acessado com o botão direito nas células.

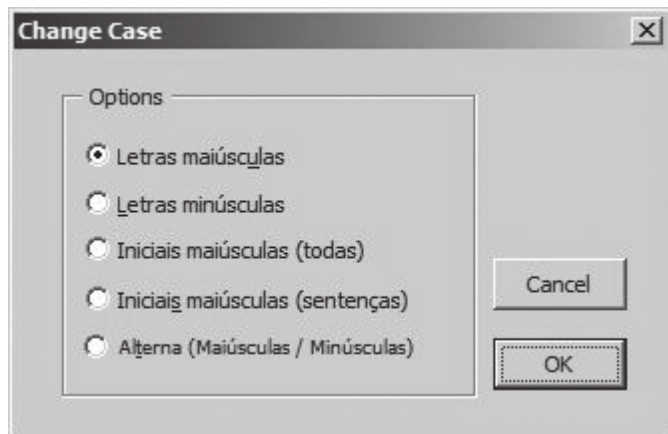
A versão original do utilitário incluía opções para letra maiúscula, letra minúscula e iniciais maiúsculas. Para a versão de add-in, acrescentei duas novas opções ao UserForm, para que ele tenha as mesmas opções que a ferramenta integrada no Microsoft Word:

5 Primeiras em maiúsculas (de sentença): Torna a primeira letra maiúscula, e todas as outras letras minúsculas.

5 Letras invertidas (alternadas): Todos os caracteres em maiúsculas são convertidos para minúsculas, e vice-versa.

ExcelVBABokks.indb 353

15/02/2013 14:00:27



354 Parte V: Juntando Tudo

A Figura 21-2 mostra o UserForm1. Os cinco botões de opção estão dentro de um controle Quadro. Além disso, o UserForm tem um botão Cancelar (chamado de CancelButton) e um botão OK (chamado de OKButton).

Figura 21-2:

O UserForm para

o add-in

Change

Case.

O código executado quando o botão Cancelar é clicado é muito simples.

Este procedimento descarrega o UserForm sem ação:

```
Private Sub CancelButton_Click()
```

```
Unload UserForm1
```

```
End Sub
```

A seguir, o código que é executado quando o botão OK é clicado. Este código faz todo o trabalho:

Private Sub OKButton_Click()

Dim TextCells As Range

Dim cell As Range

Dim Text As String

Dim i As Long

‘ Cria um objeto com constantes de texto somente

On Error Resume Next

Set TextCells = Selection._

SpecialCells(xlConstants, _

xlTextValues)

‘ Desliga a atualização de tela

Application.ScreenUpdating = False

‘ Faz o Loop pelas células

For Each cell In TextCells

Text = cell.Value

Select Case True

Case OptionLower 'lowercase

cell.Value = LCase(cell.Value)

Case OptionUpper 'UPPERCASE

cell.Value = UCase(cell.Value)

Case OptionProper 'Proper Case

cell.Value = _

WorksheetFunction.Proper(cell.Value)

ExcelVBABokks.indb 354

15/02/2013 14:00:27

Case OptionSentence 'Sentence case

Text = UCase(Left(cell.Value, 1))

Text = Text & LCase(Mid(cell.Value, 2, _

Len(cell.Value)))

cell.Value = Text

Case OptionToggle 'TOGGLE CASE

For i = 1 To Len(Text)

If Mid(Text, i, 1) Like "[A-z]" Then

Mid(Text, i, 1) = LCase(Mid(Text, i, 1))

Else

Mid(Text, i, 1) = UCase(Mid(Text, i, 1))

End

If

Next i

cell.Value = Text

End Select

Next

‘ Fecha a caixa de diálogo

Unload UserForm1

End Sub

Além das duas novas opções, esta versão do utilitário Change Case difere da versão no Capítulo 16 de outras duas maneiras:

5 Eu uso o método SpecialCells para criar um objeto variável que consiste das células na seleção que contém uma constante de texto (não uma fórmula). Essa técnica faz a rotina rodar um pouco mais depressa se a seleção contiver muitas células de fórmula. Veja o Capítulo 14 para mais informações sobre esta técnica.

5 Acrescentei o item de menu de Change Case aos menus de atalho de linha e de coluna. Assim, agora você pode executar o utilitário

clikando com o botão direito uma seleção de faixa, uma linha inteira ou um coluna inteira.

Testando a pasta de trabalho

Teste o add-in antes de converter essa pasta de trabalho. Para simular o que acontece quando a pasta de trabalho é um suplemento, você deve testá-la quando uma pasta de trabalho diferente estiver ativa. Lembre-se, um add-in nunca é uma planilha ativa ou pasta de trabalho, portanto, testá-lo quando uma diferente pasta de trabalho está aberta pode ajudá-lo a identificar alguns erros em potencial.

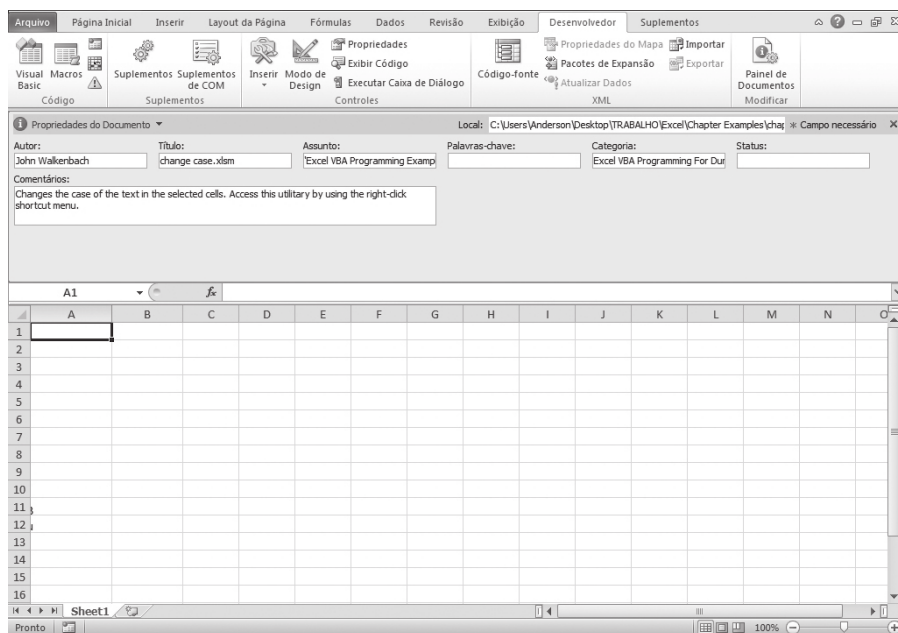
1. Abra uma nova pasta de trabalho e entre com informações em algumas células.

Com objetivos de teste, entre com vários tipos de informações, inclusive texto, valores e fórmulas. Ou apenas abra uma pasta de trabalho existente e use-a em seus testes — lembre-se de que quaisquer mudanças na pasta de trabalho não podem ser desfeitas, portanto, você pode usar uma cópia.

2. Selecione uma ou mais células (ou linhas e colunas inteiras).

ExcelVBABokks.indb 355

15/02/2013 14:00:28



3. Execute a macro ChangeCase, escolhendo o novo comando Change Case a partir do menu de atalho da célula, linha ou coluna).

Se o comando Change Case não aparecer no menu, o motivo mais

BRE-SE

provável é que você não ativou as macros quando abriu a pasta de **LEM**

trabalho change case.xlsm. Feche a pasta de trabalho e reabra-a — e assegure-se de ativar as macros.

Como adicionar informações descritivas

Eu recomendo inserir uma descrição de seu add-in, mas isso não é exigido.

1. Ative a pasta de trabalho change case.xlsm.

2. No Excel 2010, escolha Desenvolvedor⇨Painel de Documentos.

No Excel 2007, escolha Office⇨Preparar⇨Propriedades.

O Excel exibe o painel Propriedades do Documentos acima da barra de fórmula. Veja a Figura 21-3.

3. Entre com um título para o add-in (acessório) no campo Title (Título).

Esse texto aparece na lista de add-ins na caixa de diálogo Add-Ins.

Para este exemplo, entre com **Change Case**.

Figura 21-3:

Use o painel

Propriedades do

Documentos para

entrar com

informa-

ções descritivas sobre

o seu

add-in.

ExcelVBABokks.indb 356

15/02/2013 14:00:28

Capítulo 21: Criando Add-Ins do Excel

357

4. No campo Comentários, adicione uma descrição.

Essas informações aparecem embaixo da caixa de diálogo Suplementos quando o add-in é selecionado. Para este exemplo, digite **a letra do texto nas células selecionadas. Acesse este utilitário clicando com o botão direito do mouse para abrir o menu de atalho.**

Protegendo o código VBA

Se você quiser adicionar uma senha para evitar que os outros vejam o código VBA, siga estas etapas:

1. Ative o VBE e selecione a pasta de trabalho **change case.xlsm** na janela de projeto.
2. Escolha Ferramentas⇒Propriedades de VBA Project e clique na guia Proteção da caixa de diálogo que aparece.
3. Selecione a caixa de verificação Bloquear Projeto para Exibição e entre com uma senha (duas vezes).
4. Clique OK.
5. Salve a pasta de trabalho, escolhendo Arquivo⇒Salvar do menu do VBE ou voltando à janela do Excel e escolhendo Arquivo⇒Salvar.

Criando o add-in

Nesse ponto, você já testou o arquivo change case.xlsm e ele está funcionando corretamente. A próxima etapa é criar o add-in (acessório): **1. Se necessário, reative o Excel.**

2. Ative a pasta de trabalho change case.xlsm e escolha

Arquivo⇨Salvar como.

O Excel exibe a sua caixa de diálogo Salvar Como.

3. No menu drop-down Tipo, selecione Suplemento do Excel (*.xlsm).

4. Especifique o local e clique Salvar.

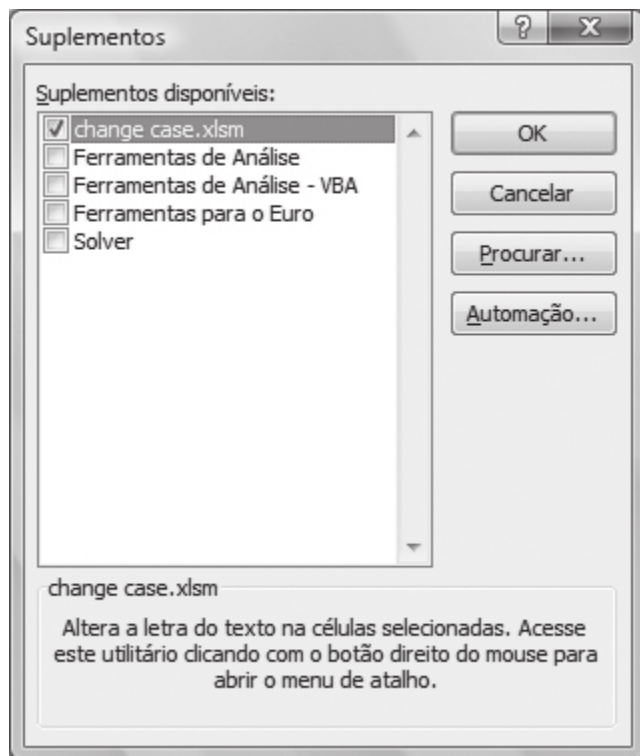
Um novo arquivo add-in (com uma extensão xlam) é criado, e a versão original em xlsm permanece aberta.

Abrindo o add-in

Para evitar confusão, feche a pasta de trabalho XLSM antes de abrir o Suplemento que você criou a partir daquela pasta de trabalho.

ExcelVBABokks.indb 357

15/02/2013 14:00:29



358 Parte V: Juntando Tudo

Abra o add-in com estas etapas:

1. Pressione Alt + MU.

O Excel exibe a caixa de diálogo Suplementos.

2. Clique no botão Procurar.

3. Localize e selecione o add-in que você acabou de criar.

4. Clique OK para fechar a caixa de diálogo Procurar.

Depois de encontrar o seu novo suplemento, a caixa de diálogo Suplementos irá listá-la. Conforme mostrado na Figura 21-4, essa caixa de diálogo também exibe as informações descritivas que você forneceu no painel Propriedades de Documentos.

5. Assegure-se de que a caixa de diálogo Add-Ins tenha uma marca de verificação para o seu novo suplemento.

6. Clique OK para fechar a caixa de diálogo.

O Excel abre o add-in e, agora, você pode usá-lo com todas as suas planilhas.

Figura 21-4:

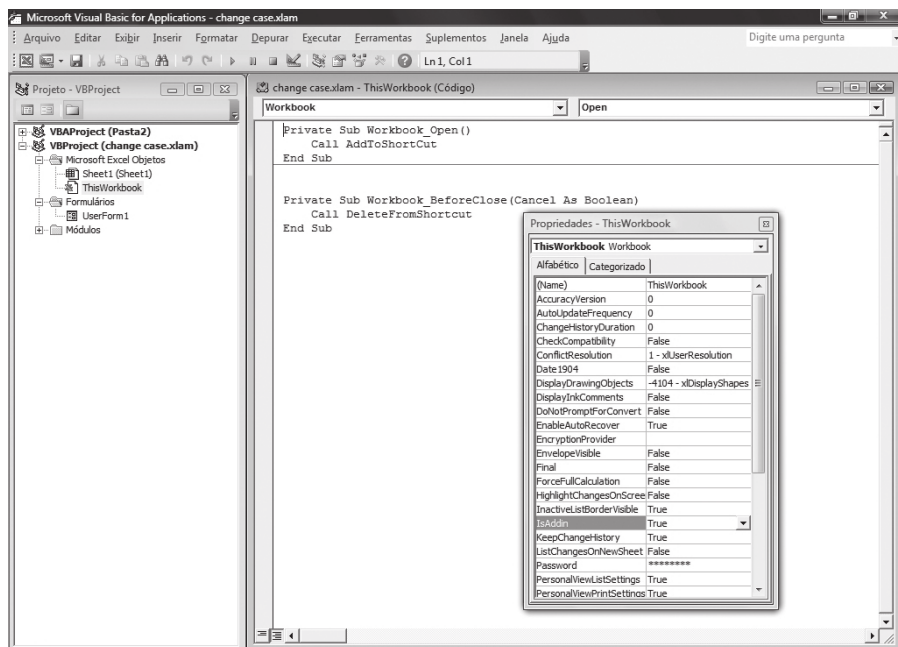
A caixa de
diálogo
Suplementos
tem o novo
suplemento
selecionado.

Distribuindo o add-in

Se você estiver se sentindo generoso, pode distribuir esse add-in para outros usuários do Excel, simplesmente dando a eles uma cópia do arquivo XLAM (eles não precisam da versão XLSM). Quando eles abrirem o suplemento, o novo comando Change Case aparecerá no menu de atalho quando eles selecionarem uma faixa, uma ou mais linhas ou colunas. Se você bloqueou o projeto VBA com uma senha, outros usuários não poderão ver o seu código de macro (a menos que eles conheçam a senha).

ExcelVBABokks.indb 358

15/02/2013 14:00:29



Capítulo 21: Criando Add-Ins do Excel

359

Como modificar o add-in

Se você já precisou modificar o acessório (e protegeu o projeto VBA com uma senha), precisa desbloqueá-lo:

1. Abra o seu arquivo XLAM se ele já não estiver aberto.
2. Ative o VBE.
3. Clique duas vezes o nome do projeto na janela de projeto.

Você é solicitado a entrar com a senha.

4. Entre com a sua senha e clique OK.

5. Faça as suas alterações no código.

6. Salve o arquivo a partir do VBE escolhendo Arquivo⇒Salvar.

Se você criar um add-in que armazena informações em uma planilha,
DICA

deve configurar a propriedade IsAddIn da pasta de trabalho para False, para ver a pasta de trabalho. Você faz isso na janela Propriedade, quando o objeto ThisWorkbook é selecionado (veja a Figura 21-5).

Depois de ter efetuado suas alterações à pasta de trabalho, assegure-se de configurar a propriedade IsAddIn de volta para True antes de salvar o arquivo.

Agora você sabe como trabalhar com add-ins e porque pode querer criar os seus próprios acessórios.

Figura 21-5:

Propriedade

InAddin do

objeto

ThisWorkbook.

ExcelVBABokks.indb 359

15/02/2013 14:00:30

360 Parte V: Juntando Tudo

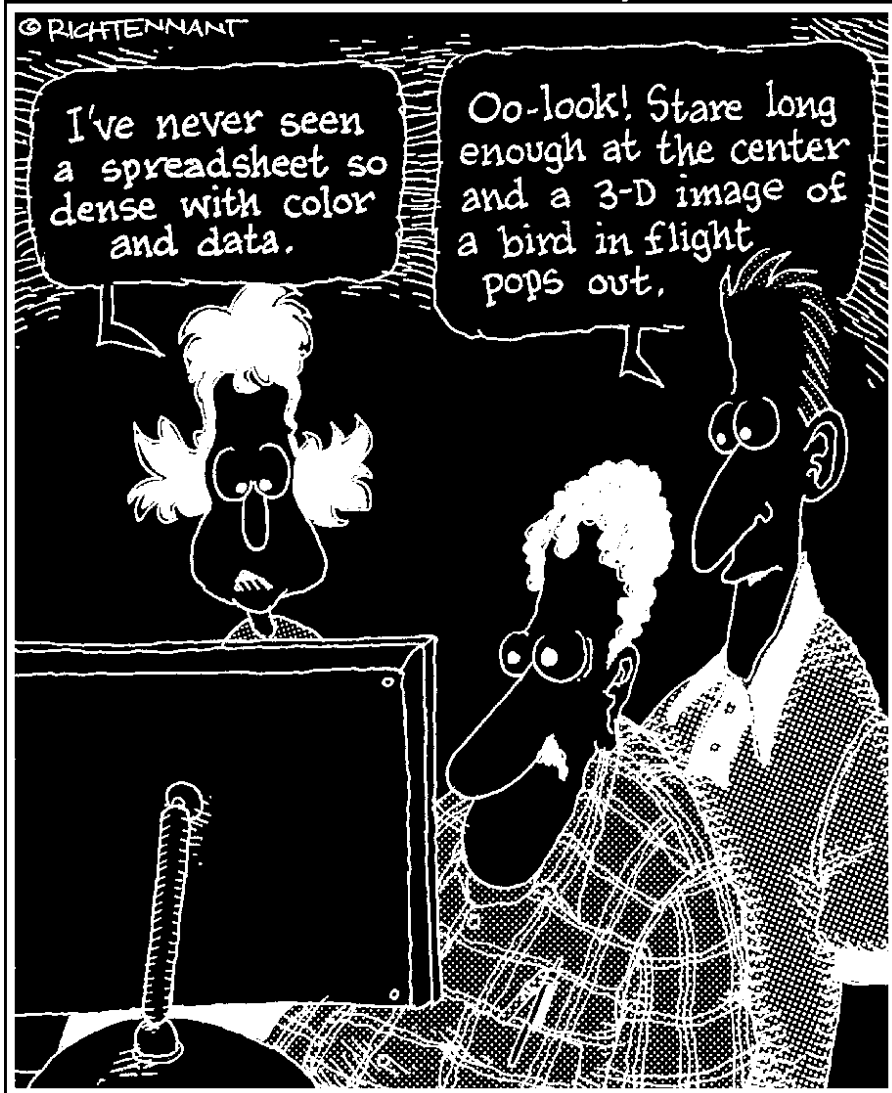
ExcelVBABokks.indb 360

15/02/2013 14:00:30

The 5th Wave

By Rich Tennant

© RICHTENNANT



Parte VI

A Parte dos Dez

A 5ª Onda

Por Rich Tennant

Eu nunca vi uma

Ó, veja! Olhe fixamente

por bastante tempo para
planilha tão cheia
o centro e verá aparecer
de cor e de dados.
uma imagem em 3D de
um pássaro
voando.

ExcelVBABokks.indb 361

15/02/2013 14:00:30

Nesta parte...

*P*or razões históricas — assim como úteis — todos
os livros da série Para Leigos têm capítulos com
listas. Os dois próximos capítulos contêm as minhas
próprias listas dos “dez”, que lidam com as perguntas
frequentes e outros recursos.

ExcelVBABokks.indb 362

15/02/2013 14:00:30

Capítulo 22

Dez Perguntas de VBA

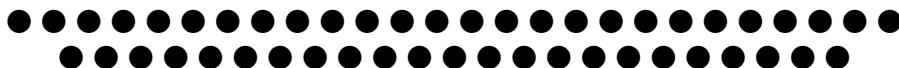
(E Respostas)



Neste Capítulo

- ☐ Armazenando procedimentos de função de planilha
- ☐ Limitações do gravador de macro

- ☐ Como aumentar a velocidade do seu código VBA
- ☐ Como declarar variáveis explicitamente
- ☐ Como usar o caractere de continuação de linha do VBA



A s seguintes dez perguntas (e respostas) tratam de algumas das questões mais comuns feitas por recém-chegados ao VBA.

Eu criei uma função VBA personalizada. Quando tento usá-la em uma fórmula, a fórmula exibe #NAME?. Qual é o erro?

Provavelmente, você colocou o seu código de função no lugar errado.

Código VBA para funções de planilha deve estar em um módulo padrão VBA — não em um módulo para uma planilha ou em ThisWorkbook. No VBE, use Inserir→Módulo para inserir um módulo padrão. Depois, corte e cole o seu código para o novo módulo VBA.

Esse é um erro muito comum, pois um módulo Sheet (folha de planilha) se parece exatamente com um módulo padrão VBA. Resista à tentação de colocar o seu código lá. Perca alguns segundos e escolha Inserir→Módulo.

Posso usar o gravador de macro do VBA para gravar todas as minhas macros?

Só se as suas macros forem bem simples. Em geral, você só o usa para gravar macros simples ou como um ponto de partida para uma macro mais complexa. O gravador de macro não grava macros que usam variáveis, loop ou qualquer outro tipo de programa com montagens em sequência. Além disso, você não pode gravar um procedimento

Function no gravador de macro do VBA. Infelizmente, o Excel 2007 se recusa a gravar muitas ações que estão relacionadas a gráficos e formas. Esse descuido foi corrigido no Excel 2010.

ExcelVBABokks.indb 363

15/02/2013 14:00:31

364 Parte VI: A Parte dos Dez

Como eu posso evitar que outros vejam o meu código VBA?

1. Ative o seu projeto no VBE e escolha Ferramentas→xxxxx Propriedades.

2. Na caixa de diálogo que aparece, clique na guia Proteção e selecione Bloquear Projeto para Exibição.

3. Entre com uma senha (duas vezes) e clique OK.

4. Salve a sua pasta de trabalho.

Fazer isso evita que usuários casuais vejam o seu código, mas com certeza, a proteção de senha não é 100% segura. Existem utilitários de quebra de senha.

Qual é o código VBA para aumentar ou diminuir o número de linhas e colunas em uma planilha?

Não existe tal código. O número de linhas e colunas é fixo e não pode ser mudado. No entanto, se você abrir uma pasta de trabalho que tenha sido criada usando uma versão anterior do Excel (antes do Excel 2007), o texto Modo de Compatibilidade aparece na barra de título. Essa informação indica que essa pasta de trabalho é limitada à antiga grade de 256-x-65536 células. Você pode se livrar dessas limitações, salvando o arquivo como uma pasta de trabalho normal (XLSX ou XLSM) e depois, fechando e reabrindo esse novo arquivo.

Quando faço referência a uma planilha em meu código VBA, recebo um erro “subscrito fora do intervalo”. Eu não estou usando quaisquer subscrições. O que há?

Esse erro acontece se você tentar acessar um elemento em uma

coleção que não existe. Por exemplo, esta declaração gera o erro se a pasta de trabalho ativa não contiver uma planilha chamada MySheet: Set X = ActiveWorkbook.Sheets(“MySheet”) No seu caso, a pasta de trabalho que você *pen*sa que está aberta, provavelmente não está (assim, não está na coleção Workbooks). Ou, talvez você tenha digitado errado o nome da pasta de trabalho.

Existe um comando VBA que seleciona uma faixa da célula ativa até a última entrada em uma coluna ou uma linha? (Em outras palavras, como uma macro pode conseguir o mesmo que Ctrl + Shift + ↓ ou Ctrl + Shift + →?) Eis o equivalente VBA a Ctrl + Shift + ↓:

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

Para outras direções, use as constantes xlToLeft, xlToRight ou xlUp ao invés de xlDown.

ExcelVBABokks.indb 364

15/02/2013 14:00:31

Capítulo 22: Dez Perguntas de VBA (E Respostas)

365

Como posso fazer o meu código VBA rodar o mais depressa possível?

Aqui estão algumas dicas:

5 Assegure-se de declarar todas as suas variáveis como um tipo específico de dados (use Option Explicit na declaração do

módulo, o que te obrigará a declarar todas as variáveis).

5 Se você fizer referência a um objeto (tal como uma faixa) mais de uma vez, crie um objeto variável usando a palavra-chave Set.

5 Sempre que possível, use a construção With-End With.

5 Se a sua macro escrever dados para uma planilha e você tiver muitas fórmulas complexas, configure o modo de cálculo para

Manual enquanto a macro rodar (mas certifique-se de fazer um cálculo quando precisar usar os resultados!).

5 Se a sua macro escrever informações para uma planilha, desative a atualização de tela, usando Application.

ScreenUpdating = False.

Não se esqueça de reintegrar essas duas últimas configurações aos seus valores iniciais quando a sua macro tiver concluído.

Como eu posso exibir múltiplas mensagens em uma caixa de mensagem?

A maneira mais fácil é montar a sua mensagem em uma string variável, usando a constante vbNewLine para indicar onde você quer que

ocorram as suas quebras de linhas. O seguinte é um rápido exemplo:
Msg = “Você selecionou o seguinte:” & vbNewLine

Msg = Msg & UserAns

MsgBox Msg

Eu escrevi algum código que apaga planilhas. Como posso evitar exibir o aviso do Excel?

Insira esta declaração antes do código que apaga as planilhas:
Application.DisplayAlerts = False

Por que não posso fazer o caractere de continuação de linha (sublinhado) do VBA funcionar?

A sequência de continuação de linha consiste, na verdade, de dois caracteres: um espaço seguido por um sublinhado. Assegure-se de usar os dois caracteres e pressionar Enter depois do sublinhado.

ExcelVBABokks.indb 365

15/02/2013 14:00:32

366 Parte VI: A Parte dos Dez

ExcelVBABokks.indb 366

15/02/2013 14:00:33

Capítulo 23

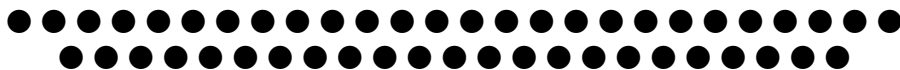
(Quase) Dez Recursos do Excel





Neste Capítulo

- ☐ Como usar o Sistema de Ajuda do VBA
- ☐ Como conseguir assistência da Microsoft
- ☐ Encontrando ajuda online



Este livro é apenas uma introdução à programação de Excel VBA. Se você estiver com fome de mais informações, fique à vontade para se alimentar com a lista de recursos adicionais que compilei aqui. Você pode descobrir novas técnicas, se comunicar com outros usuários de Excel, fazer o download de arquivos úteis, fazer perguntas, acessar a extensa Base de Conhecimentos da Microsoft e muito mais.

Vários desses recursos são serviços online ou recursos de Internet, os **BRE-SE**

quais tendem a mudar frequentemente. As descrições são precisas, **LEM**

enquanto escrevo isto, mas eu não posso garantir que essas informações permanecerão atuais. É assim que a Internet funciona.

O Sistema de Ajuda do VBA

Espero que você já tenha descoberto o sistema de Ajuda do VBA. Eu acho essa fonte de referência particularmente útil para identificar objetos, propriedades e métodos. Ela está prontamente disponível, é gratuita e (na maior parte do tempo) é preciso. Portanto, use-a.

Suporte de Produtos Microsoft

A Microsoft oferece uma ampla variedade de opções de suporte técnico (alguns gratuitos, outros por uma taxa). Para acessar os serviços de suporte da Microsoft (incluindo a útil base de conhecimento), vá para: <http://support.microsoft.com>
ExcelVBABokks.indb 367

E não se esqueça do site do Office da Microsoft, que tem quantidades de material relacionado ao Excel:

<http://office.microsoft.com>

Um outro ótimo recurso é o site Microsoft Developer Network (MSDN). Ele disponibiliza muitas, muitas informações destinadas ao desenvolvedor (é, este é você!). Aqui está um link para o site principal, onde você pode buscar por informações relacionadas ao Excel: <http://msdn.microsoft.com>

Grupos de Notícias da Internet

Os newsgroups (grupos de notícias) da Microsoft são, talvez, o melhor lugar para ir se você tem uma dúvida. Você pode encontrar centenas de grupos de notícias dedicados aos produtos da Microsoft — inclusive uma dúzia ou mais de grupos de notícias só para Excel. A melhor maneira de acessar esses grupos de notícias é usando um software especial de leitura de notícias. Ou, usar o e-mail cliente que vem com o Windows. Dependendo de sua versão do Windows, ele é chamado de Outlook Express, Windows Mail ou Windows Live Mail. Todos esses programas permitem que você se conecte com os grupos de notícias.

Configure o seu software leitor de notícias para acessar o servidor de notícias em msnews.microsoft.com.

Os grupos mais populares, em inglês, relacionados ao Excel, estão listados aqui:

microsoft.public.excel.charting

microsoft.public.excel.misc

microsoft.public.excel.printing

microsoft.public.excel.programming

microsoft.public.excel.setup

microsoft.public.excel.worksheet.functions

Se você preferir acessar os grupos de notícias usando o seu navegador da Web, há duas opções:

<http://microsoft.com/communities/>

<http://groups.google.com>

DICA

Sem ao menos saber qual é a sua pergunta, estou disposto a apostar que ela já foi respondida. Para buscar mensagens de grupos de notícias antigos por palavra-chave, aponte o seu navegador da Web para: <http://groups.google.com> ExcelVBABokks.indb 368

15/02/2013 14:00:33

Capítulo 23: (Quase) Dez Recursos do Excel 369

Sites da Internet

Vários sites contêm material relacionado ao Excel. Um bom lugar para começar a sua navegação pela Web é pelo meu próprio site, o qual é chamado de The Spreadsheet Page. Depois de chegar lá, você pode examinar o meu material e visitar a minha seção Resources (recursos), que o leva a dúzias de sites relacionados ao Excel. A URL para o meu site é: <http://spreadsheetpage.com>

Blogs do Excel

Você pode encontrar, literalmente, milhões de blogs na Web. Um *blog* é, basicamente, um diário atualizado frequentemente sobre um assunto em especial. Muitos blogs são dedicados exclusivamente ao Excel. Eu mantenho uma lista de blogs de Excel em meu site: <http://spreadsheetpage.com/index.php/excelfeeds>

Google

Quando eu tenho uma dúvida sobre qualquer assunto (inclusive programação do Excel), a minha primeira linha de ataque é o Google — atualmente, o site de busca mais popular do mundo.

<http://google.com>

Insira alguns termos chave de busca e veja o que o Google encontra. Eu consigo uma resposta em cerca de 90% das vezes. Se isso falhar, então eu procuro nos grupos de notícias (descritos anteriormente), usando esta URL: <http://groups.google.com>

Bing

Bing é a resposta da Microsoft ao site de busca Google. Algumas pessoas o preferem ao Google; outras não. Se você não tiver experimentado, a URL é:

<http://bing.com>

ExcelVBABokks.indb 369

15/02/2013 14:00:33

370 Parte VI: A Parte dos Dez

Grupos e Usuários Locais

Muitas comunidades maiores e universidades têm um grupo de usuários de Excel que se encontra periodicamente. Se você puder encontrar um grupo de usuários em sua área, verifique-o. Geralmente, esses grupos são uma excelente fonte de contatos e compartilhamento de ideias.

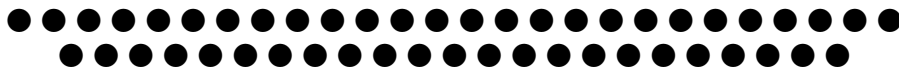
Meus Outros Livros

Sinto muito, mas eu não pude resistir à oportunidade de uma propaganda descarada. Para levar a programação de VBA ao próximo nível, dê uma olhada no meu *Excel 2007 Power Programming with VBA* ou no *Excel 2010 Power Programming with VBA* (ambos publicados pela Wiley).

ExcelVBABokks.indb 370

15/02/2013 14:00:33

Índice



• Símbolos •

planilha, funções de, 350

Power Utility Pak, 349

= (sinal de igual), 109-110

shareware, 349

‘ (apóstrofe), em comentários, 95-96

Solver, 349

+ (adição), operador de, 110-111

Suplementos (add-ins)

/ (divisão), operador de, 110-111

terceirizados, fornecedores, 349

^ (exponenciação), operador de, 110-111

AddinUninstall, evento de, 162

\ (divisão de inteiro), operador de, 110-111

Address (endereço), propriedade de

* (multiplicação), operador de, 110-111

(Objeto Range), 122

& (concatenação), operador de, 110

vantagens do VBA, 17

- (subtração), operador de, 110-111

estética de UserForms, 287

adição (+), operador de, 110-111

adicionando. Ver também inserindo

• A •

À faixa de Opções, 16

controles aos UserForms, 254, 256-257,

aumentando número de fileiras/colunas em

261-263, 269-270

uma planilha, 364

alerta, mensagens de, 228-229

abrindo add-ins (acessórios), 351-352

alertas

Abs (absoluto), função, 133

desativando avisos de alerta, 365

absolute (absoluto), modo (gravador de

alerta de segurança, 32

macro), 84-85

alinhando controles em UserForms, 284-285

Accelerator (Acelerador), propriedade, 263,

Alterando

271

configurações, 220-222

aceleração, teclas de, 263, 285-286

controle, propriedades de, 257-258,

Activate (ativar), evento, 64, 162-163,

270-272

170-174

ordem de guia em caixas de diálogo,

AddinInstall, evento, 162

285-286

abrindo, 351-352

alterando propriedades, 62-63

Analysis ToolPak (Pacote de Ferramen—

alternando entre janelas, 43

tas de Análise), 349

aplicativos

carregando, 351

criação de, 16

Change Case (Alterar/Mudar Letra),

usuários de, CB 20

suplemento, 353-359

apóstrofo (') em comentários, 95-96

código, 350

Application, objeto, 56

convertendo arquivos de pasta de

argumentos em funções

trabalho em um add-in (acessório),

argumento opcional, 340-342

350

definidos, 335

criando/criação, 16, 349-353

descrições, 347-348

definido, 349

dois argumentos, 338-339

descarregando, 351

faixa de argumento, 339-340

distribuindo/distribuição, 358

número indefinido de argumentos,

Faixa de Opções, 352

342-343

fechando, 351

sem argumento, 336

IsAddin, propriedade, 349

um argumento, 336-338

modificando, 359

armazenando macros, 26-27, 88

Nota: *CB (Capítulo de Bonificação) em um número de página, significa que o conteúdo está localizado no site deste livro (www.altabooks.com.br) — procurar pelo nome do livro).*

ExcelVBABokks.indb 371

15/02/2013 14:00:33

372 Programando Excel VBA Para Leigos, Tradução da 2a Edição
Array, função, 133

bugs de versão errada, 198

arrays

bugs. Ver também depuração

declarando, 112

atualizações de segurança, 198

definidos, 112

bugs de casos extremos, 197

dinâmicos, 113-114

bugs de texto incorreto, 197

funções de, 343-345

bugs de tipo errado de dados, 198

multidimensionais, 113

bugs de versão errada, 198

Asc, função, 133

comentários, 208

ativando VBE, 37

definidos, 183

Atn, função, 133

dicas para redução, 208-209

atualização de tela, 227-228

erros de sintaxe, 198

Auto Indent (Recuo Automático), opção

falhas lógicas, 197

(VBE), 52

identificando, 198-199

Auto listar membros, opção (VBE), 51, 66-67

mensagens de erro, 198-100

Auto verificar sintaxe, opção (VBE), 50

procedimentos Function (função), 209

automático, cálculo, 228

procedimentos Sub, 209

automatizando/automatizar

definição, 14

operações repetitivas, 15

• C •

tarefas, 15

AutoSize, propriedade, 271

Caixa de seleção, controle, 256, 274

avisos de alerta, 365

caixas de diálogo personalizadas. Ver caixas
de diálogo

caixas de mensagem, 238-241

• B •

calculando pagamento de hipoteca, 137

Calculate (calcular), evento, 163

BackColor (cor de fundo), propriedade, 271

cálculo automático, 228

BackStyle (estilo de fundo), 271

caminhando através do código, 205

backups, CB 20

Caption (legenda), propriedade, 272

Banfield, Colin, Excel 2010 For Dummies

carregando add-ins (acessórios), 351

Quick Reference, 2

Cells, propriedade, 117-118

barras de ferramentas

Célula, menu de atalho de, 326-328

personalizando, 329-330

células, faixas de. Ver faixas

apagando, 330

chamando um procedimento Function

VBE, 38-39

(função) de um procedimento Sub, 79

barras de ferramentas, 329-330

Change Case, suplemento, 353-359

BeforeClose, evento, 162, 169

Change, evento, 163, 175-177

BeforeDoubleClick, evento, 163, 174

Choose, função, 133

BeforePrint (antes de imprimir), evento, 162

Chr, função, 133

BeforeRightClick (antes de clicar com botão

circulando através de janelas, 43

direito), evento, 163, 174-175

Clear (limpar), método, 128

BeforeSave, evento, 162, 169-170

Code (código), janela

Bing, 369

em VBE, 38-39, 42-44

blogs, 369

para UserForms, 258

Booleano, configurações, 221

Code Colors (código de cores), opção

Booleano, tipo de dados, 99

(VBE), 52-53

botões

código

Acesso rápido, barra de ferramentas,

add-ins (acessórios), 350

320

breakpoints (pontos de interrupção),

acrescentando à faixa de opções, 16

202-203

botões personalizados, 16

caminhando através de linha por linha,

macros, 76-77

205

botões personalizados, 16

código modular, BC18

Break, modo, 203-207

comentários, 95-97

bugs de tipo de dados errado, 198

convenções usadas neste livro, 4

ExcelVBABokks.indb 372

15/02/2013 14:00:33

Índice 373

copiando, 49

objetivo (em), 106

entrando em módulos VBA, 44-49

valores, 105-107

etiquetas, 113

constantes integradas, 106-107

falhas lógicas, 197

contadores, 99

limpando, BC18

contratando projetos, 40

números de linha, 113

controles (UserForms)

recuando, 52, 208

acrescentando/adicionando, 254,

rodando tão rápido quanto possível,

256-257, 261-263, 269-270

365

ajuda, 273

senha de proteção, 364, BC17-BC18

alinhando, 284-285

sequência de caractere de continuação

CheckBox (caixa de verificação), 256

de linha, 4-5

ComboBox (caixa combinada), 256

variáveis, 97-98

CommandButton (botão de comando),

versões de Excel, CB20

257, 275-276

colando faixas, 127

controle CheckBox (caixa de verifica—

coleções

ção), 274

definidas, 19, 58

controle ComboBox (caixa combinada),

loops, 158-160

274-275

métodos, 64

controle Frame (moldura), 276

propriedades, 63

controle ListBox (caixa de lista),

Color (cor), propriedade, 125

296-301

comandos personalizados, 15

controle OptionButton (botão de

ComboBox (caixa combinada), controle,

opção), 302-303

256, 174-175

controle RefEdit, 280-281, 301-302

comentários

controle SpinButton (botão de rotação),

apóstrofe (‘), 95-96

303-305

bugs, 208

controle TextBox (caixa de texto),

convertendo declarações em comentá-

303-305

rios, 96

cores, 271

definidos, 95

dimensionando, 270

inserindo, 95-96

espaçando, 284-285

uso efetivo de, 97

Frame (moldura), 256

CommandBars (barras de comando),

Image (imagem), 257, 276-277

objeto, 250-252, 313, 321-325

imagens, 272

CommandButton (botão de comando),

Label (etiqueta), 256, 277

controle, 257, 275-276

ListBox (caixa de lista), 256, 278-279

compilação, erros de, 97

movendo, 284

comprimento de string, 130-131

mudando propriedades, 257-258,

computador, vírus de, 5-6

270-272

concatenação, 110

MultiPage (múltiplas páginas), 257, 279

configurações. Ver também propriedades

nomeação, 259-260, 272

configurações Booleanas, 221

ocultando, 284

mudando, 220-222

OptionButton (botão de opção), 256,

configurações não Booleanas, 221-222

279-280

segurança, 5

posicionando, 270

configurando pontos de interrupção no

propriedade Accelerator (acelerador),

código, 202-203

271

Const (constante), declaração, 105

propriedade AutoSize (tamanho

constantes

automático), 271

constantes integradas, 106-107

propriedade BackColor (cor de fundo),

constantes pré-definidas para cores,

271

CB1

propriedade BackStyle (estilo de

declaração Const, 105

fundo), 271

declarando, 105

propriedade Caption (legenda), 272

função MsgBox (caixa de mensagem),

propriedade Left and Top (esquerda e

238-239

superior/no alto), 272

ExcelVBABokks.indb 373

15/02/2013 14:00:33

374 Programando Excel VBA Para Leigos, Tradução da 2a Edição

propriedade Name (nome), 272

aplicativos, 16

propriedade Picture (imagem, retrato),

botões, 16

caixas de diálogo, 254-255, 260-263

propriedade Value (valor), 272

caixas de mensagem, 236-237

propriedade Visible (visível), 272

comandos, 15

propriedade Width and Height (largura

expressões de observação, 206-207

e altura), 272

funções de planilha, 333-335

propriedades, 257-258, 270-272

funções, 16

redimensionando, 284

macros, 25-27

RefEdit, 257

módulos VBA, 44

ScrollBar (barra de rolagem), 257, 281

UserForms (formulários de usuário),

SpinButton (botão de rotação), 257, 282

254-255, 260-263, 290-292

TabStrip, 257, 282

Ctrl + Chift + ↓ e Ctrl + Shift + →, 364-365

TextBox (caixa de texto), 256, 282-283

CurDir, função, 133

ToggleButton (botão de alternância),

Currency (moeda), tipo de dados, 99

256, 283

CurrentRegion (região atual), propriedade,

controles (UserForms), 284

214

controles(CommandBars), 323-325

custom UI Editor for Microsoft Office

convenções, explicadas, 4

(Editor de Interface de Usuário Persona—

convertendo

lizada para Microsoft Office), 317-320

arquivos de pasta de trabalho em

add-ins (acessórios), 350

cores, CB3-CB4, CB6-CB7

• *D* •

declarações em comentários, 96

cópia, proteção, 21

dados, procedimento de cópia de, BC20

copiando

dados, tipos de

código, 49

Booleano, 99

faixas, 127, 212-214

bugs de tipo errado de dados, 198

Copy (copiar), método, 127

Currency (moeda), 99

cores

Date (data), 99, 108

Color (cor), propriedade, 125

definidos por User (usuário), 99

constantes pré-definidas, BC1

definidos, 98

controles (UserForms), 271

Double [duplo] (negativo), 99

convertendo, BC3-BC4, BC6-BC7

Double [duplo] (positivo), 99

escala cinza, BC5-BC8

identificando tipo de dados de um

especificando, BC1

objeto selecionado, 132

objetos, 125

Integer (inteiro), 99

sistema de cor RGB, CB1-CB2, CB8-CB9

integrados, 99

temas, CB10-CB15

Long (longo), 99

ThemeColor (cor de tema), proprieda—

Object (objeto), 99

de, CB12

Single (único), 99

TintAndShade (matiz e tonalidade),

String, 99

propriedade, 125, BC12-BC13

Variant (variante), 99

valor decimais de cor, CB1-CB5

variáveis, 98-99, 230

Column (coluna), propriedade (objeto

dados, tipos de Date (data), 99, 108

Range [faixa]), 121-122

dados, validação de, 175-177, 295

colunas

data, exibindo, 130

aumentando o número de, 364

Date (data), função, 130, 133

diminuindo o número de, 364

DateAdd (acrescentar data), função, 133

objetos Range (faixa), 119

DateDiff (data diferente), função, 133

cores, BC1

DatePart (data em especial), função, 133

Cos (cosseno), função, 133

DateSerial (data em série), função, 133

Count (contar), propriedade (objeto Range

DateValue (valor de data), função, 133

[faixa]), 121

Day (dia), função, 133

criando

Deactivate (desativar), função, 162-163,

add-ins (acessórios), 16, 349-353

170-174

ExcelVBABokks.indb 374

15/02/2013 14:00:34

Índice 375

Debug.Print (depurar impressão), declara—

exemplo de, 289-293, 295-296

ções, 201-202

exibindo caixas de diálogo integradas,

decimais, valor de cor, CB1-CB5

250-251

DECIMAL2RGB, função, CB3-CB4

exibindo, 259, 265, 292

decisão, tomada de, construções. Ver

indicador de progresso, 305-308

programa, fluxo de

inserindo, 255-256

declarações

janela Code (código), 258

declarações de designação, 109-110

lista de verificação para criação,

convertendo em comentários, 96

311-312

definição, 45

macros, 265-268, 292-293

declarações em módulos VBA, 44

movendo controles, 284

declarando

mudando propriedades de controle,

arrays, 112

257-258, 270-272

constantes, 105

nomeando controles, 259-260

tipos de dados variáveis, 230

ocultando controles, 284

variáveis, 98-105, BC17

ordem de tabulação, 285-286

Default to Full Module View (padronizar

posicionando controles, 270

para exibição módulo completo), opção

procedimentos que lidam com eventos,

(VBE), 52

263-265, 294-295

Delete (apagar), método, 128

quando usar, 253-254

depurando

redimensionando controles, 284

declarações Debug-Print (depurar

tabuladas, 308-310

impressão), 201-202

teclas de acelerador, 285-286

definição, 198

testando, 287

função MsgBox (caixa de mensagem),

usuários de teclado, 285

200-201

validação de dados, 295

pontos de interrupção, 202

Dicas Automáticas de Dados, opção (VBE),

procedimentos Function (função), 341

51

técnicas de, 199-202

dicas de velocidade, 227-229, 231

VBA Debugger (depurador VBA),

Dim, 102-104

202-208

Dim, declaração, 102-104

desativando, 328

Dim, palavra-chave, 102-104

descarregando add-ins (acessórios), 351

dimensionando controles (UserForms), 270

descrições

diminuindo o número de fileiras/colunas em

de argumentos, 347-348

uma planilha, 364

de funções, 346-347

dinâmicos, arrays, 113-114

de macros, 26, 88

Dir, função, 133

designando

distribuindo add-ins (acessórios), 358

macros a um botão, 76-77

divisão (/), operador de, 110-111

valores a variáveis, 20

DoUntil (fazer até), loops, 142, 157-158

designando declarações, 109-110

DoWhile (fazer enquanto), loops, 142,

desvantagens de VBA, 17

156-157

determinando um tamanho de arquivo,

Docking (acoplar), guia (VBE), 54

131data, 130

docking (acoplar), janela de, 54

Developer (desenvolvedor), guia, 23-24

documento, temas de

diálogo, caixas de. Ver também UserForms

cores, BC10-BC15

acrescentando controles a, 254, 256-257,

definição, BC10

261-263, 269-270

fazendo download, BC10

alinhando controles, 284-285

efeitos, BC10

alternativas a, 235-236

fontes, BC10

criação de, 254-255, 260-263, 290, 292

misturando e combinando elementos,

de gráficos, 310-311

BC10

dimensionando controles, 270

Double [duplo] (negativo), tipo de dados,

espaçando controles, 284-285

estética de, 287

Double [duplo] (positivo), tipo de dados, 99

ExcelVBABokks.indb 375

15/02/2013 14:00:34

376 Programando Excel VBA Para Leigos, Tradução da 2a Edição

Drag-and-Drop Text Editing (arrastar e

procedimento EnterSquareRoot (entrar

soltar de edição de texto), opção (VBE),

com raiz quadrada), 184-188

52

erros de tempo de execução, 182

especificando

• E •

etiquetas, 113

eventos

Activate (ativar), 64, 162-163, 170-174

editando

AddInInstall (instalar acessório), 162

macros, 30

AddInUninstall (desinstalar acessório),

módulos VBA, 18

162

Editor Format (editor de formatação), guia

BeforeClose (antes de fechar), 162, 169

(VBE), 52-53

BeforeDoubleClick (antes de clicar duas

vezes), 163, 174

BeforePrint (antes de imprimir), 162

BeforeRightClick (antes de clicar com o

botão direito), 163, 174-175

BeforeSave (antes de salvar), 162,

End (fim/final/término/encerramento),

169-170

declaração, 70

Calculate (calcular), 163

End Function (término de função), declara—

Change (mudar/alterar), 163, 175-177

ção, 70

Deactivate (desativar), 162-163, 170-174

End Sub, declaração, 70

definidos, 64, 161-162

EntireColumn (coluna inteira), propriedade,

FollowHyperlink (seguir hyperlink), 163

hora/horário, de, 177-180

EntireRow (fileira inteira), propriedade, 215

macros, 163

entrando (com/inserindo)

NewSheet (nova folha de planilha), 162

código em módulos VBA, 44-49

OnKey (na tecla), 180-181

funções de planilha, 138

OnTime (na hora), 178-180

eqv (equivalência), operador de, 111

Open (abrir), 162, 167-168

Erl, função, 133

pastas de trabalho, em, 162

Err (erro), objeto, 193-194

planilhas, em, 163

Err, função, 133

SelectionChange (mudar seleção), 163

erro “assinatura fora de faixa”, 364

SheetActivate (ativar planilha), 162

erro, mensagens de, 198-199

SheetBeforeDoubleClick (planilha antes

Error (erro), função, 133

de clicar duas vezes), 162

erros

SheetBeforeRightClick (planilha antes

bugs, 183

de clicar com o botão direito), 162

declaração Exit Function (sair de

SheetCalculate (calcular planilha), 162

função), 192

SheetChange (alterar planilha), 162

declaração Exit Sub (sair de Sub), 192

SheetDeactivate (desativar planilha),

declaração On Error Resume Next (em

162

erro, ir para próxima), 192

SheetFollowHyperlink (planilha

declaração On Error, 189-190, 192

seguindo o hyperlink), 162

declaração Resume (ir; continuar),

SheetSelectionChange (alterar seleção

190-192

de planilha, 162

erro “subscript out of range” (assinatu—

teclas de pressionar, de, 177, 180-181

ra fora de faixa), 364

WindowActivate (ativar janela), 162

erros de compilação, 97

WindowDeactivate (desativar janela),

erros de programação, 183

162

erros de tempo de execução, 183

WindowResize (redimensionar janela),

identificando, 193-194

162

ignorando, 192-193

eventos, procedimentos que lidam com

intencionais, 194-196

programação, 164-167

macros, 22

UserForms, 263-265, 294-295

objeto Err, 193-194

Excel

ExcelVBABokks.indb 376

15/02/2013 14:00:34

Índice 377

considerando outras opções, BC19

janelas, 44

funções de planilha, 135-139

Ferramentas de análise, 349

mudando configurações, em, 220-222

FileDialog (diálogo de arquivo), objeto, 249

Office Compatibility Pak (pacote de
fileiras

compatibilidade Office), 22

diminuindo o número de fileiras, 364

versões de, 20-22, 198, CB20

aumentando o número de fileiras, 364

Excel 2007 Power Programming with VBA

objetos Range (faixa), 119

(Walkenbach), 370

FileLen (extensão de arquivo), função,

Excel 2010 Bible (Walkenbach), 2

131-132, 134

Excel 2010 For Dummies (Harvey), 2

Fix (corrigir), função, 131, 134

Excel 2010 For Dummies Quick Reference

fixo, comprimento de strings, 107-108

(Walkenbach e Bandfield), 2

FollowHyperlink (acompanhar hyperlink),

Excel 2010 Power Programming with VBA

evento de, 163

Walkenbach), 370

Font (fonte), opção (VBE), 53

Excel, modelo objeto, 55

Font (fonte), propriedade (objeto Range

Excel, programação

[faixa]), 123

benefícios e aprender, 3

fontes (temas), BC10

código, 4

For-Next (para próximo), loops, 142, 152-156

definição, 2

forçando modo Break (pausa/interrupção),

exemplo de arquivos, 8

204

sequência de caractere de continuação

Format (formatar/formato), função, 134

de linha, 4-5

formatando gráficos, 225-226

vírus, 5-6

Forms (formulários), nó de, 41

executando

Formula (fórmula), propriedade (objeto

procedimentos Function (função), 70,

Range [faixa]), 124

78-80

Frame (moldura), controle, 256, 276

procedimentos Sub, 70-78

funções (planilha)

ExecuteMso (execute Mso), método,

add-ins (acessórios), 350

250-251

argumentos, 335-343

exemplo de arquivos, 8

criação, 333-335

exemplos, 326-327

DECIMAL2RGB, BC3-BC4

exibindo, data ou horário do sistema, 130

descrições, 346-347

Exit Function (sair de função), declaração,

entrando (com), 138

192

exemplos, 336-341

Exit Sub (sair de Sub), declaração), 192

LARGE (grande), 136

Exp (expandir), função, 133

MAX (máximo), 136

expandindo projetos, 40

personalizadas, 333-335

experimentando / testando, BC19

PMT, 137

exponenciação (^), operador de, 110-111

RGB (vermelho, verde, preto), 125, 134,

exportando objetos, 42

CB2

expressões

RGB2DECIMAL, CB3-CB4

bugs de situações extremas, 197

VLOOKUP, 137-138

caractere de continuação de linha, 109

WorksheetFunction (função de plani—
definição, 109

lha), objeto, 139

expressões watch (de observação),

funções (VBA)

206-207

Abs (absoluto), 133

funções, 109

Ano, 135

extraindo variáveis da memória, 105

argumentos, 335-336

Array, 133

• *F* •

arrays, 343-345

Asc, 133

Atn, 133

faixas, 128

Choose (escolher), 133

fechando

Chr, 133

add-ins (acessórios), 351

Cos, 133

ExcelVBABokks.indb 377

15/02/2013 14:00:34

378 Programando Excel VBA Para Leigos, Tradução da 2a Edição
criação de, 16

Replace (substituir), 135

CurDir, 133

Right (à direita), 135

Date (data), 130, 133

Rnd (random – aleatório), 135

DateAdd (adicionar data), 133

RTrim, 135

DateDiff (data diferente), 133

Second (segundo), 135

DatePart (parte de data), 133

Sgn, 135

DateSerial (data em série), 133

Shell (envoltório), 132, 135

DateValue (valor de data), 133

Sin, 135

Day (dia), 133

Space (espaço), 135

definidas, 129, 334

Split (separar), 135

dia da semana, 135

Sqr (quadrada), 135

Dir, 133

Str (rua), 135

Erl, 133

StrComp (complemento de rua), 135

Err, 133

String, 135

Error (erro), 133

Tan (tangente), 135

Exp, 133

Time (horário), 135

expressões, 109

Timer (temporizador), 135

FileLen (extensão de arquivo), 131-132, 134

TimeSerial (horário em série), 135

Fix (corrigir), 131, 134

TimeValue (valor de horário), 135

Format (formatar/formato), 134

Trim (apagar), 135

GetSetting (obter configuração), 134

TypeName (digitar nome), 132, 135

Hex (hexagonal), 134

UBound (limite de maiúsculas), 135

Hour (hora), 134

UCase (letras maiúsculas), 135, 253

InputBox (caixa de entrada), 132, 134,

User Defined Function (UDF) [Função

218, 241-244

Definida por Usuário], 138

InStr, 134

Val, 135

Int, 131, 134

VarType (tipo de variável), 135

integradas, 130-135

funções integradas, 130-135

Ipmt, 134

funções personalizadas, 16, 138-139, 333-335

IsArray (é array), 134

Function (função), 70

IsDate (é data), 134

Function (função), procedimentos

IsEmpty (está vazio), 134

bugs, 209

IsError (é erro), 134

chamando a partir de um procedimento

IsMissing (está faltando), 134

Sub, 79

IsNull (é nulo), 134

declaração End Function (fim/término

IsNumeric (é numérico), 134

de função), 70

IsObject (é objeto), 134

definidos, 18, 69-70

LBound (limite à esquerda), 134

depurando, 341

LCase (tipo de letra à esquerda), 134

executando, 70, 78-80

Left (esquerdo), 134

gravando, 70

Len (comprimento/extensão), 130-131,

módulos VBA, 44

134

nomeando, 71

Log (registro), 134

palavra-chave de Function (função), 70

lookup (consultar), 137-138

palavras reservadas, 98

LTrim (apapar à esquerda), 134

Mid (meio), 134

MIN (mínimo), 136

• G •

Minute (minuto), 134

Month (mês), 134

General (geral), guia (VBE), 53-54

MsgBox (caixa de mensagem), 132, 134,

GetOpenFilename (obter nome de arquivo

200-201, 236-241

aberto), método, 244-248

MultiplyTwo (multiplicar por dois),

GetSaveAsFilename (obter nome de arquivo

138-139

salvado como), método, 148-249

Now (agora), 134

GetSetting (obter configuração), função, 134

personalizadas, 16, 138-139, 333

ExcelVBABokks.indb 378

15/02/2013 14:00:34

Índice 379

globais, variáveis, 104-105

ícones, explicados, 8

Google, 369

identificando

GoTo (ir para), declaração, 142-143

bugs, 198-199

grade de linhas, 270

erros, 193-194

gráficos

tipo de dados de um objeto seleciona—

Chart, propriedade, 224

do, 132

ChartObject (objeto gráfico), objeto,

If-Then (se, então), estrutura, 142-148

223-224

If-Then-Else (se, então, outra coisa), 144-145

ChartObjects (objetos gráficos),

ignorando erros, 192-193

coleção, 224-225

Image (imagem), controle, 257, 276-277

escala/cinza, deC7-bC8

imagens para controles de UserForm, 272

formatando, 225-226

Immediate (imediata), janela (VBE), 38-39,

gráficos embutidos, 223-224

204-205

macros, 222-223

imp (importação), operador de, 111

modificando tipo de gráfico, 224

importando objetos, 42

propriedades, 225

incorreto, bugs de contexto, 197

selecionando, 226

InputBox (caixa de entrada), função, 132,

UserForms, 310-311

134, 218, 241-244

gravando macros

Insert Function (inserir função), caixa de

eficiência, 90-92

diálogo, 345-348

gravador de macro, 26-27, 30, 47-48,

inserindo. Ver também adicionando/

81-92, 363

acrescentando

modo absoluto, 84-85

comentários, 95-96

modo relativo, 85-86

funções de planilha em um módulo

módulos VBA, 47-48

VBA, 138

o que é gravado, 87-88

texto, 15

opções de gravação, 88-89

UserForms, 255-256

orientações passo a passo, 26-27, 82-83

InStr, função, 134

preparação, 25, 84

Int, função, 131, 134

procedimentos Function (função), 70

Integer (inteiro), tipo de dados, 99

procedimentos Sub, 70

integer, operador de divisão (\), 110

GUI (Graphical User Interface – Interface

inteiro, parte de um número, exibindo, 131

Gráfica de Usuário), 255

intencionais, erros, 194-196

Interior, propriedade (objeto Range [faixa]),

• *H* •

123-124

Internet, grupos de notícias, 368

IPmt, função, 134

habilitando macros, 5-6, BC19

IsAddin (é acessório), propriedade, 349

Harvey, Greg, Excel 2010 For Dummies, 2

IsDate (é data), função, 134

HasFormula (tem fórmula), propriedade

IsEmpty (está vazio), função, 134

(objeto Range [faixa]), 122-123

IsError (é erro), função, 134

help (ajuda)

IsMissing (está faltando), função, 134

controles (Userforms), 273

IsNull (é nulo), função, 134

Sistema de Ajuda VBA, 367

IsNumeric (é numérico), função, 134

Hex (hexagonal), função, 134

IsObject (é objeto), função, 134

hierarquia de objetos, 18-19, 56-57

Hora, 130

hora, eventos, 177-180

• *L* •

hora, exibindo, 130

Hora, função de, 134

Label (etiqueta), controle, 256, 277

LARGE (grande), planilha, função de, 136

• *I* •

LBound, função, 134

LCase (letra minúscula), função, 134

Left (esquerda), função, 134

Informações rápidas automáticas, opção

Left and Top (esquerda e no alto), proprie-

(VBE), 51

dade, 272

ExcelVBABokks.indb 379

15/02/2013 14:00:34

380 Programando Excel VBA Para Leigos, Tradução da 2a Edição

Len (extensão), função, 130-131, 134

Ctrl + Shift + ↓, 364-365

limpando código, 18

definição, 14

limpando faixas, 128

desativando, 5-6

linha, continuação de, caractere de, 109

descrição, 26

linha, números de, em código, 113

designando a um botão, 76-77

linha, sequência de caractere de continua—

editando, 30

ção de, 4-5, 365

erros, 22

lista de verificação para criar UserForms,

eventos, 163

311-312

gráficos, 222-223

lista, retornando, 344-345

gravando, 26-27, 30, 47-48, 81-92, 363

listando menus de atalho, 321-322

habilitando, 5-6, BC19

listando, 321-322

macro de NameAndTime (nome e hora),

ListBox (caixa de lista), controle, 256,

25-33

278-279, 296-301

nomeando, 26, 88

locais confiáveis, 32

salvando, 30-31

locals (locais), janela (VBE), 207-208

segurança, 31-32

Log (registro), função, 134

teclas de atalho, 26, 75-76, 88

lógicas, falhas, em código, 197

testando, 27, 267-268

lógicos, operadores, 111

UserForms, 265-268, 292-293

Long (longos), tipo de dados, 99

vendo, 27-29

lookup (consulta), funções, 137-138

macros, 5-6

loop, contadores de, 99

Margin Indicator Bar (margem de barra de

loops

indicação), opção (VBE), 53

coleções de, 158-160

MAX (máximo), função de planilha, 136

definidas, 151-152

maximizando janelas, 42-43

faixas, 216-218

maximizando, 42-43

loops DoUntil (fazer até), 142, 157-158

memória, 105

loops DoWhile (fazer enquanto), 142,

mensagem, caixas de

156-157

criação, 236-237

loops For-Next (para próximo), 142, 152,

exibindo o valor de duas variáveis,

154-156

200-201

velocidade de, 146

mensagens de múltiplas linhas, 365

LTrim, função, 134

personalizando, 238-241

respostas, 237-238

• *M* •

mensagens de alerta, 228-229

menu de atalho Cell (célula), 326-328

menu, barra de, em Visual BasicEditor

macro, gravador de

(VBE), 38

eficiência, 90-92

menus de atalho, 321-328

limites de, 83-84

menus de atalho, 328

modo absoluto, 84-85

métodos

modo relativo, 85-86

Auto List Members (lista automática de

módulos VBA, 47-48

membros), 66-67

o que é gravado, 87-88

coleções, 64

opções de gravação, 88-89

definidos, 20, 60-61, 63-64

orientações passo a passo, 26-27, 82-83

Delete (apagar), 128

preparação para usar, 25, 84

especificando, 20

procedimentos Function (Função), 70

ExecuteMso, 250-251

procedimentos Sub, 70

GetOpenFilename (obter nome de

macros

arquivo aberto), 244-248

armazenando, 26-27, 88

GetSaveAsFilename (obter nome de

barra de ferramentas Quick Access

arquivo salvado como), 248-249

(acesso rápido), 266

método Clear (limpar), 128

criando, 25-27

método Copy (copiar), 127

ExcelVBABokks.indb 380

15/02/2013 14:00:34

Índice 381

método Paste (colar), 127

• N •

método Select (selecionar), 126-127

objeto Range (faixa), 126

#NAME, fórmula, 363

objetos Range (faixa), 127-128

Name (nome), propriedade, 272

Sistema Help (de ajuda) VBA, 65

NameAndTime (nome e hora), macro

métodos, 20

armazenando, 26-27

Microsoft Developer Network Desenvolve—

criação, 25-27

dor de Rede Microsoft), site (MSDN),

descrição, 26

368

editando, 30

Microsoft Excel

gravando, 26-27, 30

considerando outras opções, BC19

nomeando, 26

funções de planilha, 135-139

recursos, 25-33

mudando configurações, 220-222

salvando, 30-31

Office Compatibility Pak (pacote de

segurança, 31-32

compatibilidade Office), 22

tecla de atalho, 26

versões, 20-22, 198, BC20

testando, 27

Microsoft Excel 2010 Power Programming

vendo, 27-29

with VBA (Walkenbach), 2

nomeando

Microsoft Excel Objects, nó, 40

controles (UserForms), 259-260, 272

Microsoft Product Support (suporte de

macros, 26, 88

produtos Microsoft), 367-368

procedimentos Function (função), 71

Mid (meio), função, 134

procedimentos Sub, 71

MIN (mínimo), função de planilha, 136

variáveis, 97-98

minimizando janelas, 42-44

NewSheet (nova folha de planilha), evento,

Minute (minuto), função, 134

162

misturando e combinando elementos de

nós (projetos)

tema de documento, BC10

definidos, 40

modificando

nó de Forms (formulários), 41

add-ins (acessórios), 359

nó de Microsoft Excel Objects, 40

tipo de gráfico, 224

nó de Modules (módulos), 41

modular, código, BC18

não booleanos, configurações, 221-222

Modules (módulos), nó, 41

não contínuas, faixas, 116-117

módulo aritmético, operador (mod), 110

not (não), operador, 111

módulo VBA a um projeto, 41

Now (agora), função, 134

módulo, variáveis apenas de, 103-104

NumberFormat (formato de número),

módulos. Ver módulos VBA

propriedade (objeto Range [faixa]), 126

Month (mês), função, 134

movendo

controles (UserForms), 284

• O •

faixas, 215

MSDN (site Microsoft Developer Network),

Object Browser (objeto navegador), 66

368

Object Model Map (mapa modelo objeto),

MsgBox (caixa de mensagem), função, 132,

57

134, 200-201, 236-241

objeto CommandBars (barras de comando),

multidimensionais, arrays, 113

321-325

MultiPage (múltiplas páginas), controle,

ocultando, 43-44

257, 279

organizando, 43

múltiplas linhas, mensagens de, 365

Objeto, tipo de dados de, 99

multiplicação, operador (*), 110-111

Objetos Ativos, 19

MultiplyTwo (multiplicar dois), função

objetos. Ver também objetos por nome

personalizada, 138-139

ativos, 19

MultiSelect (múltipla seleção), propriedade,

coleções, 19, 58

299-300

cores, 125

definição, 18, 55-56

ExcelVBABokks.indb 381

15/02/2013 14:00:34

382 Programando Excel VBA Para Leigos, Tradução da 2a Edição
eventos, 64

• P •

exportando, 42

expressões, 109-110

parênteses, usando para mudar o operador

hierarquia, 18-19, 56-57

de ordem de prioridade, 111

identificando tipo de um objeto

parte inteira de um número, 131

selecionado, 132

Paste (colar), método, 127

importando, 42

personalizando, 321-328

métodos, 20, 60-61, 63-64

Picture (imagem, retrato), propriedade, 272

navegando através da hierarquia, 59

planilhas, 365

Object Browser (objeto navegador), 66

planilhas, funções. Ver funções (worksheet

Object Model Map (mapa modelo

= planilha)

objeto), 57

PMT, função de planilha, 137

propriedades, 19, 60-63

Pontos de interrupção

referências, 19, 58-60, 229-230

configurando, 202-203

VBA Help System (sistema de ajuda do

definidos, 202

VBA), 65

removendo, 203

Office Compatibility Pak (pacote de

posicionando controles em UserForms, 270

compatibilidade Office), 22

Power Utility Pak, 349

Offset (desvio), propriedade, 118-119

pré-definidas, constantes para cores, BC1

On Error (em erro), declaração, 189-190, 192

prioridade, ordem de para operadores, 111

On Error Resume Next (em erro ir para

procedimento, apenas, variáveis de, 102-103

próximo), declaração, 192-193, 208

procedimentos. Ver procedimentos que

OnKey (na tecla), evento, 180-181

lidam com eventos; procedimentos

OnTime (na hora), evento, 178-180

Function (função); procedimentos Sub

OOB (programação orientada a objeto), 55

Procedure Separator (procedimento

Open (abrir), evento, 162, 167-168

separador), opção (VBE), 52

operador (divisão de inteiro), 110

produto, ativação, 21

operadores

programa, fluxo de

adição (+), 110-111

declaração GoTo (ir para), 142-143

concatenação de string (&), 110

descrição, 141

divisão (/), 110-111

estrutura ElseIf (e se), 146-147

divisão de inteiro (\), 110 e, 111

estrutura If-Then (se, então), 142-148

eqv (equivalência), 111

estrutura If-Then-Else (se, então, outra

exponenciação (^), 110-111

coisa), 144-145

imp (importando), 111

estrutura Select Case (selecionar tipo

lógicos, 111

de letra), 142, 148-151

módulo aritmético (mod), 110

estrutura With-End With, 231

multiplicação (*), 110-111

loops DoUntil (fazer até), 142, 157-158

not (não), 111

loops DoWhile (fazer enquanto), 142,

or (ou), 111

156-157

ordem de prioridade, 111

loops For-Next (para próximo), 142,

sinal de igual (=), operador de, 109-110

152-156

subtração (-), 110-111

programação

usando parênteses para mudar ordem

benefícios de aprender, 3

de prioridade, 111

código, 4

XoR, 111

definição, 2

Option Explicit (opção explícita), declara—

exemplo de arquivos, 8

ção, 100-101, 208

experimentando, BC19

OptionButton (botão de opção), controle,

programação estruturada, 143

256, 279-280, 302-303

sequência de caractere de continuação

or (ou), operador, 111

de linha, 4-5

organizando módulos VBA, 44

vírus, 5-6

orientado a objeto, programação (OOP), 55

programação estruturada, 143

programação, erros de, 183

programas, bugs de. Ver bugs

ExcelVBABokks.indb 382

15/02/2013 14:00:34

Índice 383

programas, definidos, 14

propriedade Column (coluna), 121-122

progresso, indicador de, 305-308

propriedade Count (contar/contagem),

Project (projeto), janela (VBE), 28, 38-42

121

projetos

propriedade CurrentRegion (região

acrescentando um módulo VBA, 41

atual), 214

contratando, 40

propriedade EntireColumn (coluna

definidos, 40

inteira), 215

expandindo, 40

propriedade EntireRow (fileira inteira),

nós, 40-41

215

removendo um módulo VBA, 41-42

propriedade Font (fonte), 123

Properties (propriedades), janela (VBE), 38

propriedade Formula (fórmula), 124

propriedades. Ver também propriedades

propriedade HasFormula (tem fórmula),

por nome

122-123

Auto List Members (lista automática de

propriedade Interior (interna), 123-124

membros), 66-67

propriedade NumberFormat (formato

coleções, 63

de número), 126

controles (UserForms), 257-258, 270

propriedade Offset (descio), 118-119

definição, 19, 60-62

propriedade Row (fileira), 121-122

gráficos, 225

propriedade Text (texto), 121

mudando, 62-63

propriedade ThemeColor (cor de tema),

objetos Range (faixa), 119-126

125

propriedade ThemeColor (cor de tema),

propriedade Value (valor), 120

125

propriedades apenas de leitura, 119

referências, 19

propriedades, 119-126

Sistema de Ajuda VBA, 65

referências, 115-119

vendo, 62-63

selecionando, 211

protegendo

Range (faixa), palavra-chave, 115-117

código, 364, BC17-BC18

ranges (faixas). Ver também Range (faixa),

regras de validação de dados, 177

objetos

protegendo código, 364, BC17-BC18

apagando, 128

Public (pública), declaração, 104

argumentos, 339-340

Public (pública), palavra-chave, 104

colando, 127

Public (público), 104

copiando, 127, 212-214

públicas, variáveis, 104-105

determinando tipo de seleção, 219

faixas de tamanhos variáveis, 213-214

Q

faixas nomeadas, 212

limpando, 128

loops, 216-218

Quick Access (acesso rápido), barra de

movendo, 215

ferramentas, 266, 320

múltiplas faixas, 219-220

selecionando, 126-127, 211-212, 214-215,

R

219-220

solicitando por um valor de célula,

Range (faixa), objetos. Ver também ranges

218-219

(faixas)

valor máximo em uma faixa, 136

colunas, 119

ReDim, declaração, 113-114

definição, 115

redimensionando

faixas descontínuas, 116-117

controles (UserForms), 284

fileiras, 119

janelas, 43

método SpecialCells (células especiais),

redimensionando, 43

216-217

Redo (refazer), botão (VBE), 45-46

métodos, 126-128

RefEdit, controle, 257, 280-281, 301-302

propriedade Address (endereço), 122

referências

propriedade Cells (células), 117-118

a objetos Range (faixa), 115-119

propriedade Color (cor), 125

a objetos, 19, 58-60, 229-230

ExcelVBABokks.indb 383

15/02/2013 14:00:34

384 Programando Excel VBA Para Leigos, Tradução da 2a Edição
as propriedades de objeto, 19

segurança, BC17-BC18

referência totalmente qualificada, 59

selecionando

relativo, modo (gravador de macro), 85-86

arquivos múltiplos, 247-248

removendo. Ver também apagando

faixas, 126-127, 211-212, 214-215, 219-220

barras de ferramentas, 330

gráficos, 226

módulo VBA de um projeto, 41-42

objetos Range (faixa), 211

pontos de interrupção, 203

selecionando múltiplos arquivos, 247-248,

reorganizando janelas, 43

Select (selecionar), método, 126-127

reorganizando, 43

Select Case (selecionar tipo de letra),
repetitivas, operações, automatizando, 15
estrutura, 142, 148-151

Replace (substituir), função, 135

SelectionChange (mudança de seleção),

Require Variable Declaration (exige declara—
evento, 163

ção e variável), opção (VBE), 50

separadoras, barras, 52

reservadas, palavras

Sgn, função, 135

procedimentos Function (função), 98

shareware, 349

procedimentos Sub, 98

SheetActivate (ativar planilhas), evento, 162
variáveis, 98

SheetBeforeDoubleClick (planilha antes de

Resume (retomar; continuar), declaração,
clicar duas vezes), evento, 162

190-192

SheetBeforeRightClick (planilha antes de
retornando uma lista classificada, 344-345
clicar com o botão direito), evento, 162

RGB (vermelho, verde, preto), sistema de

SheetCalculate (calcular planilha), evento,

cor, BC1-BC2, BC8-BC9

162

RGB, função, 125, 134, BC2

SheetChange (alterar planilha), evento, 162

RGB2DECIMAL, função, BC3-BC4

SheetDeactivate (desativar planilha),

Ribbon (barra gráfica)

evento, 162

add-ins (acessórios), 352

SheetFollowHyperlink (planilha seguindo

adicionando botões, 16

hyperlink), 162

guia Developer (desenvolvedor), 23-24

SheetSelectionChange (alteração de seleção

personalizando, 24, 314-321

de planilha), evento, 162

Ribbon (barra gráfica), 24, 314-321

Shell (envoltório), função, 132, 135

Right (direita), função, 135

Sin, função, 135

Rnd (random = aleatório), 135

sinal de igual (=), operador de, 109-110

rodando

Single (único), tipo de dados, 99

código o mais rápido possível, 365

sintaxe, erros de, 198

Visual Basic Editor (VBE), 37

Size (tamanho), opção (VBE), 53

rotinas, definidas, 45

Solver (solucionador), 349

Row (fileira), propriedade (objeto Range

Space (espaço), função, 135

[faixa]), 121-122

SpecialCells (células especiais), método,

RTrim, função, 135

216-217

SpinButton (botão de giro), controle, 257,

• S •

282, 303-305

Split (separação), função, 135

Spreadsheet Page (página de planilha

salvando macros, 30-31

financeira), A, Web site, 369

ScrollBar (barra de rolagem), controle, 257,

Sqr (quadrada), função, 135

281

Stop (parar), 204

Second (segunda), função, 135

Stop (pare/parar), palavra-chave, 204

segurança

Str, função, 135

configurações, 5

StrComp, função, 135

locais confiáveis, 32

String, função, 135

macros, 31-32

string, operador (&) de concatenação, 110

senhas, BC17-BC18

String, tipo de dados, 99

Trust Center (centro de confiança), 5

strings

segurança, alerta de, 32

comprimento fixo, 107-108

segurança, atualizações, 198

comprimento variável, 107

ExcelVBABokks.indb 384

15/02/2013 14:00:34

Índice 385

concatenação, 110

TimeSerial (horário em série), função, 135

definidas, 107

TimeValue (valor de hora), função, 135

descobrimo, extensão de string,

TintAndShade (tonalidade e matiz),

130-131

propriedade, 125, BC12-BC13

Sub, 70

tipos de dados integrados, 99

Sub, procedimentos

ToggleButton (botão de alternância),

bugs, 209

controle, 256, 283

chamando um procedimento Function

toques de tecla, 177, 180-181

(função), 79

totalmente qualificada, referência, 59

declaração End Sub, 70

Trim (aparar), função, 135

definidos, 18, 69-70

trocando entre janelas, 43

executando, 70-78

Trust Center (centro de confiança), 5

gravando, 70

TypeName (digitar nome), função, 132, 135

módulos VBA, 44

nomeando, 71

palavra-chave Sub, 70

• U •

palavras reservadas, 98

subtração, operador (-), 110-111

UBound, função, 135

UCase, função, 135, 253

UDF (User Defined Function – função

• T •

definida por usuário), 138

Undo (desfazer), botão em VBE, 45-46

tab (guia), ordem em caixas de diálogo,

User Defined Function (UDF – função

285-286

definida por usuário), 138

TabStrip (separação de tabulação), função,

UserForms (formulários de usuário),

135

250-251, 259, 265, 292

tabulação, caixas de diálogo, 308-310

UserForms. Ver também caixas de diálogo

tarefas, automatizando, 15

acrescentando controles, 254, 256-257,
tecla, combinações de, 4
261-263, 269-270
teclas de atalho para macros, 16, 75-76, 88
alinhando controles, 284-285
tema, cores de, 125
alternativas a, 235-236
temas
criando, 254-255, 260-263, 290-292
de cores, BC10-BC15
dimensionando controles, 270
definidos, BC10
espaçando controles, 284-285
efeitos, BC10
estética, 287
fazendo download, BC10
exemplo, 289-293, 295-296
fontes, BC10
exibindo, 250-251, 259, 265, 292
misturando e combinando elementos
gráficos, 310-311
de, BC10
indicador de progresso, 305-308
terceiros, fornecedores de add-ins (acessó-

inserindo, 255-256

rios), 349

janela Code (código), 258

testando

lista de verificação para criação,

macros, 27, 267-268

311-312

UserForms, 287, 293

macros, 265-268, 292-293

Text (texto), propriedade (objeto Range

movendo controles, 284

[faixa]), 121

mudando propriedades de controle,

TextBox (caixa de texto), controle, 256,

257-258, 270-272

282-283, 303-305

nomeando controles, 259-260

texto

ocultando controles, 284

inserindo, 15

ordem de guia/tabulação, 285-286

strings, 107

posicionando controles, 270

ThemeColor (cor de tema), propriedade

procedimentos que lidam com eventos,

(objeto Range [faixa]), 125, BC12

263-265, 294-295

Time (hora), função, 135

quando usar, 253-254

Timer (temporizador), função, 135

redimensionando controles, 284

ExcelVBABokks.indb 385

15/02/2013 14:00:34

386 Programando Excel VBA Para Leigos, Tradução da 2a Edição
tabulado, 308-310

valores, 105

teclas de aceleração, 285-286

vida de (duração), 105

testando, 287, 293

variáveis estáticas, 104-105

usando informações a partir de, 259

variáveis, 99-105

usuários de teclado, 285

variáveis, dimensionadas, faixas de, 213-214

validação de dados, 295

variável, extensão de, strings, 107

UserName (nome de usuário), propriedade,

VarType (tipo de variante), função, 135

VBA (Visual Basic for Applications)

Usuário, tipo de dados definidos por, 99

benefícios de aprender, 3

usuários de teclado e caixas de diálogo, 285

código, 4

usuários, BC20

definição, 13

usuários, grupos de, 370

desvantagens de, 17

exemplo de arquivos, 8

• V •

sequência de caractere de continuação

de linha, 4-5, 365

usos comuns, 14-16

Val (validação), função, 135

vantagens de, 17

validação, regras de, 175-177

vírus, 5-6

valor máximo em uma faixa, 136

VBA, código. Ver código

valores

VBA, Debugger (depurador), 202-208

constantes, 105-107

VBA, funções. Ver funções (VBA)

designando a variáveis, 20

VBA, gravador de macro. Ver gravador de
encontrando valor máximo em uma
macro

faixa, 136

VBA, Help System (sistema de ajuda), 65,
solicitando o valor de uma célula,
367

218-219

VBA, módulos

valores decimais de cor, BC1-BC5

acrescentando a um projeto, 41

variáveis, 105

copiando código, 49

Value (valor), propriedades

criando, 44

controles (UserForms), 272

declarações, 44

objeto Range (faixa), 120

definição, 334

Variant (variante), tipos de dados, 99

editando, 18

variáveis

entrando com código, 44-49

apenas módulo, 103-104

inserindo funções de planilha, 138

apenas procedimento, 102-103

organizando, 44

arrays, 112-114

procedimentos Function (função), 18,

declaração Dim, 102-104

44

declaração End (fim/final), 105

procedimentos Sub, 18, 44

declaração Option Explicit (opção

removendo de um projeto, 41-42

explícita), 100-101

vendo, 18

declaração Public (pública), 104

VBE (Visual Basic Editor). Ver Visual Basic

declarando, 98-105, BC17

Editor (VBE)

definição, 20, 97

velocidade de loops, 146

designando valores, 20

vendo

estáticas, 104-105

macros, 27-29

exibindo o valor de duas variáveis,
propriedades, 62-63
200-201
módulos VBA, 18
expressões, 109-110
versões de Excel, 20-22, 198, BC20
globais, 104-105
vida/duração de variáveis, 105
nomeação, 97-98
vírus, 5-6
objetivo, 99-105
Visible (visível), propriedade, 272
palavras reservadas, 98
Visual Basic Editor (VBE)
públicas, 104-105
ativando, 37
tipos de dados, 98-99, 230
barra de menu, 38
tirando da memória, 105
barras de ferramentas, 38-39
ExcelVBABokks.indb 386
15/02/2013 14:00:34

botão Redo (refazer), 45-46

Excel 2010 Bible,

botão Undo (desfazer), 45-46

Excel 2010 For Dummies Quick Referen—

guia Docking, 54

ce, 2

guia Editor Format (editor de formata—

Excel 2010 Power Programming with

ção), 52-53

VBA, 370

guia Editor, 50

Microsoft Excel 2010 Power Program—

guia General (geral), 53-54

ming with VBA, 2

janela Code (código), 38-39, 42-44

watch (observar), expressões de, 206-207

janela Immediate (imediata), 38-39,

Watches, janela (VBE), 206-207

204-205

Web sites

janela Locals (locais), 207-208

Microsoft Developer Network, site

janela Project (projeto), 28, 38-42

(MSDN), 368

janela Properties (propriedades), 38

Microsoft Product Support, 367-368

janela Watches, 206-207

Spreadsheet Page (página de planilha),

menus de atalho, 39

The (a), 369

novos recursos, 40

Weekday (dia da semana), função, 135

opção Auto Data Tips (dicas de dados),

Width and Height (largura e altura),

51

propriedade, 272

opção Auto Indent (recoo automático),

WindowActivate (ativar janela), evento, 162

52

WindowResize (redimensionar janela),

opção Auto List Members (lista

evento, 162

automática de membros), 51, 66-67

Window Deactivate (desativar janela), 162

opção Auto Quick Info (informações

Window Resize (redimensionar janela), 162

rápidas automáticas), 51

With-End With, estrutura, 231

opção Auto Syntax Check (verificação

workbooks (pastas de trabalho)

automática de sintaxe), 50

convertendo arquivos de pasta de

opção Code Colors (código de cores),

trabalho em add-ins (acessórios),

52-53

350

opção Default to Full Module View

eventos, 162

(padrão para visualizar módulo

WorkRange (faixa de trabalho), objeto, 218

inteiro), 52

WorksheetFunction (função de planilha),

opção Drag-and-Drop Text Editing

objeto, 136-139

(edição de texto de agarrar e soltar),

worksheets (planilhas)

52

apagando, 365

opção Font (fonte), 53

eventos, 163

opção Margin Indicator Bar (barra

indicadora de margem), 53

opção Procedure Separator, 52

• X •

opção Require Variable Declaration

(exige declaração de variável), 50

XLAM, extensão de arquivo, 350

opção Size (tamanho), 53

XoR, operador, 111

personalizando, 49-54

rodando, 37

Visual Basic Editor (VBE), 49-54

Visual Basic for Applications (VBA). Ver

• Y •

VBA Visual Basic for Applications)

Year (ano), função, 135

VLOOKUP, função de planilha, 137-138

• W •

Walkenbach, John

Excel 2007 Power Programming with

VBA. 370

ExcelVBABokks.indb 387

15/02/2013 14:00:35

388 Programando Excel VBA Para Leigos, Tradução da 2a Edição
ExcelVBABokks.indb 388

15/02/2013 14:00:35

Document Outline

- [CapaExcelVBA](#)
- [MioloExcelVBAParaLeigos](#)